DESIGN AND IMPLEMENTATION OF AN INTELLIGENT PRIMITIVE DRIVER

By

PETER M. VINCH, JR.

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2003

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE **2003**	2. REPORT TYPE	3. DATES COVERED **00-00-2003 to 00-00-2003**

4. TITLE AND SUBTITLE **Design and Implementation of an Intelligent Primitive Driver**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Center for Intelligent Machines and Robotics,Department of Mechanical Engineering,University of Florida,Gainesville,FL,32611**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**84**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

“What we call the beginning is often the end. And to make an end is to make a beginning. The end is where we start from.”

T.S. Eliot, “Four Quartets”

ACKNOWLEDGMENTS

There are many individuals, on professional and personal fronts, whom I would like to thank for making possible the preparation and completion of this thesis. Professionally, I would like to thank Dr. Carl D. Crane, III (my academic advisor and chair of my graduate committee) for his direction, advice and encouragement. I also greatly appreciate the time and energy that the other members of my supervisory committee (Dr. John K. Schueller and Dr. Eric M. Scwartz) dedicated to my cause. I would like to extend my gratitude to the Nuclear and Radiological Engineering Department of the University of Florida for allowing me the use of their Remotec Andros robot for testing purposes. I thank the U.S. Department of Energy (Grant #DE-FG04-86NE37967) and the Air Force Research Laboratory located at Tyndall Air Force Base, Florida (contract # F08637-00-C6008) for all of their support. I am also grateful to the entire staff of the Center for Intelligent Machines and Robotics, for all of their assistance with the completion of my thesis. Additionally, I would like to distinguish the contributions of Shannon Ridgeway, Daniel Kent, Carl P. Evans, III, and especially Dr. David Keith Novick for all of their professional advice and the friendship that they have extended to me. Without these individuals, none of this would have been possible.

Personally, I would like to thank all of my friends and family for their efforts to keep me sane and functional. Key people in this effort were Andrew and Gretchen McGraw, whose friendship, support and love I will always value. I would also like to thank Adam Feinberg for being such a great roommate, lifting partner, and best friend. He has pushed me to become better in every aspect of my life; and for this I will always be grateful. Of course, without the love and support of my family, all of my efforts would be fruitless. Therefore, I would like to thank my mother, Nancy; my two sisters, Bridget and Melissa; and my grandfather, Philip. Finally, I would like to thank my father, Peter M. Vinch, Sr.; and my uncle, Robert Frie. These men have been my greatest influences, helping to place me on the proper path through life. Without them, I would have been lost.

TABLE OF CONTENTS

APPENDIX

LIST OF TABLES

LIST OF FIGURES

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

DESIGN AND IMPLEMENTATION OF AN INTELLIGENT PRIMITIVE DRIVER

By

Peter M. Vinch, Jr.

August 2003

Chair: Carl D. Crane, III
Department: Mechanical and Aerospace Engineering

An Intelligent Primitive Driver (IPD) was designed to supplement the control of a Primitive Driver component that is defined in the Department of Defense Joint Architecture for Unmanned Systems (JAUS). Whereas the Primitive Driver component accepts and blindly executes wrench commands, the IPD uses various subsystems to provide it with the necessary information to make low-level decisions concerning vehicle control. The IPD is accessible by either an onboard autonomous control system; or by a tele-operational control system. Tele-operational control (teleop) is characterized by the direct control of a platform by a human operator. For the case of an autonomous control system, the IPD reduces high-level control responsibilities; and therefore reduces processor demands. In the case of teleop control, the IPD serves to ease operator burden by automating intensive operator-controlled processes.

The test platform for the functionality of the Intelligent Primitive Driver was a Remotec ANDROS robot. In the case of the ANDROS robot, the IPD automates the process of maneuvering up or down a flight of stairs.

CHAPTER 1
INTRODUCTION AND BACKGROUND

1.1 Introduction

The purpose of this research was to design and develop an experimental intelligence component based on the Joint Architecture for Unmanned Systems (JAUS), currently being developed in concert with the Department of Defense. This component will be designated the Intelligent Primitive Driver (IPD) and will allow for assisted tele-operational (teleop) control of a platform. Teleop control can be defined as the control of a vehicle or robot by the direct input of a human operator. Assisted teleop is based upon this concept, but includes additional computer control to supplement the operator's control; intensive control procedures were automated to ease the burden on the operator. The IPD accomplished this task by having direct control of any and all actuators that directly affect the motion of the platform. A Remotec Andros robot, outfitted with a JAUS-compliant controller, was used to test the viability of the IPD. In the case of the Andros, the IPD automated the complicated stair ascending and descending operations.

1.2 Background

1.2.1 Assisted Teleop Control

The human-machine relationship has always been clearly defined: machines are controlled by humans and are constructed to ease the burdens of human life. However, two recently developing factors have begun to dramatically change this relationship.

First, an increase in computing power and system reliability along with a decrease in size and power requirements of modern control systems have made possible the shift of some control responsibility from the human operator to the controller itself. Bayouth and colleagues [1] (of the Robotics Institute of Carnegie Mellon University) constructed an autonomous roadway vehicle that demonstrated lane following, speed and heading compliance, and obstacle avoidance in an effort to increase vehicle safety and mobility. Several other groups have also worked on the "Intelligent Cruise Control" concept. The Mustererkennung und Szenenanalyse Pattern Recognition and Scene Analysis (MESA) research group [2] have developed behaviors (such as tracking a lead vehicle or lane changing) that comprise a basic control set on which more complicated procedures are based. Schlegel [3], in his thesis, "Autonomous Vehicle Control using Image Processing," tested the intelligent cruise control notion through the use of scale models coupled with a remote control station aimed at emulating full-scale vehicle controls.

The second factor that has influenced the human-machine interface is the growing complexity of the platforms being automated and the tasks they are able to perform. Connell and Viola [4] (of the IBM T.J. Watson Research Center) have a novel way of describing the problems with traditional platform control systems:

> "Consider the differences between riding a horse and driving an automobile. A horse will not run into a telephone pole at high speed. If you fall asleep in the saddle, a horse will continue to follow the path that it is on.… In general, horses are much smarter than automobiles and thus provide a better model for control of a mobile robot." [4]

Connell and Viola constructed a platform and control system that cast the operator as an agent in the control system, able to input commands into the arbitration network, as well as a high-level command language that was able to shut out individual control agents.

Huntsberger and Rose [5], of the Jet Propulsion Laboratory (JPL) and the University of South Carolina (USC), respectively, are working on a behavior-based control system named BISMARC, or the Biologically Inspired System for Map-based Autonomous Rover Control, for use with the Mars rover platforms. They utilize a form of path planning that is derived from the study of human navigation through complex environments.

1.2.2 Stair Climbing

For mobile robotic platforms to be useful in urban environments, they need to be able to easily traverse commonly found terrain features, such as stairways. Several detection methods are used to enable platform navigation on these non-continuous planar structures (such as vision-based positioning and edge detection, laser-based edge detection, and the fusion of several sensor systems). Others have solved the dilemma by manipulating the physical characteristics of the platform to suite the special needs of a stair-climbing vehicle.

Matthies, et al. [6] constructed a small mobile platform capable of performing reconnaissance duties in urban situations. To meet the stair climbing requirements associated with this detail, a vision-guided navigation and detection system was implemented on the robot. A stereovision system was used to detect the forward edges of individual steps. This information was used to derive the angle of rotation between the robot and the stairway; and to ensure that the robot was accurately following the stair heading.

Lewis and Simo' [7] (Iguana Robotics, Inc.) built a bipedal platform based on the biomorphic concept capable of stair travel (Figure 1-1). Basically, a biomorphic robot attempts to mimic the sensory capabilities of animals; sensory input is predicated upon voluntary movement of the platform and must be distinguished from possible inputs produced by the platform itself. The platform fused stereovision; and tactile and pressure sensors to build an accurate data model of the terrain to traverse.

Another application of the biomorphic concept is the work of Talebi, et al. [8]. They constructed a quadruped platform that has only one actuator per leg coupled with compliant prismatic joints. Therefore, as an animal would, the platform climbs a stair dynamically. As the leg contacts the stair, ground forces cause the joint spring to compress; and the effective length of the leg is reduced.

Perhaps the most stable solution for the stair-climbing problem, when the development of the platform permits, is to tailor the geometry to accommodate the stair climbing motion. Lauria, et al. [9] took this approach in developing their vehicle, *Octopus*. The platform consists of eight motorized, tactile wheels and a tilt sensor; and has a total of fifteen degrees of freedom. The platform geometry allows for all eight wheels to be in contact with the ground at all times, regardless of the terrain profile; allowing for relatively simple travel of the platform across any uneven terrain, stairways included.

1.3 The Joint Architecture for Unmanned Systems (JAUS)

This section provides a functional description of the Joint Architecture for Unmanned Systems (JAUS). The technical constraints on the architecture, system topology, standard component definition, and the JAUS message will be discussed.

1.3.1 Overview

The JAUS architecture is being developed in conjunction with the Department of Defense in support of unmanned vehicle systems development and provides a means for reducing system life-cycle costs by offering a well-defined component interface. This interface allows for the future reuse of expensive components in developing autonomous systems and also allows for the quick exchange of malfunctioning or outdated components in current autonomous systems. Further, the engineer is free to place all available resources into obtaining optimal performance from the component as its interface is predefined [10].

The JAUS architecture is divided into three separate volumes: the JAUS Domain Model, Document Control Plan, and Reference Architecture. The JAUS Domain Model defines both known and prospective operational requirements of unmanned systems, while the Document Control Plan describes the procedure used to recognize and track changes to accepted JAUS documents. This work will focus solely upon the Reference Architecture as it is concerned with aspects of component design. The reference architecture defines components, messages and standards that classify a system paradigm, which allows for the assimilation of distributed system architectures. The reference architecture is comprised only of components and messages that have been technically evaluated by the tech base, academia, or an industry source and whose implementation is thoroughly understood. Therefore, the incorporation of components, their classification, and their corresponding messages is an evolutionary process [10].

There are four technical constraints, which have been levied upon the JAUS architecture to ensure that it may be freely applied to any classification of unmanned system. These constraints are platform independence, mission isolation, computer hardware independence, and technological independence. To be platform independent, no assumptions concerning vehicle platforms to be automated will be incorporated into the definition of any JAUS component. The JAUS architecture defines a mission as the ability to gather information about or to alter the state of the environment in which the platform is operating. Therefore, the mission isolation constraint is intended to allow the engineer to construct systems that can support a variety of missions, not just a single one. The computer hardware independence constraint insists that the component be designed independent of the computer system upon which the component will run. This was included in the architecture so that currently used components can be applied to future computer systems without modification to the component. This allows new computer architectures to be easily inserted into current systems, extending its life cycle. This also allows for hardware flexibility, as the appropriate hardware may be applied to each system. Finally, technological independence is similar to computer hardware independence, but instead deals with the action to be performed by the component instead of the system that is performing the action. This constraint basically states that the architecture will make no assumptions concerning the method by which an action is performed. For example, in a JAUS compliant position system, no assumptions are made concerning the method in which the vehicle position is obtained. Any method of obtaining vehicle position is allowable, from a Global Positioning System (GPS), dead reckoning, inertial measurement, a vision based position system, or any subsequent positioning system [10].

1.3.2 System Topology

There are four elements, which comprise the hierarchy of the JAUS architecture: the System, Subsystem, Node, and Component/Instance. A System is a logical grouping of one or more Subsystems, which have been grouped such that beneficial cooperation between the Subsystems can be achieved. A Subsystem is a distinct organism, which is comprised of any number of Nodes necessary to form a complete unmanned system. A

Node is a distinct entity, comprised of a single processor or multiple processors working in conjunction with each other, to provide a complete service. Finally, a Component is a cohesive software process, which runs on a Node. An Instance is a single occurrence of a Component running on a Node. Several Instances of the same Component may run on a single Node, and are delineated by unique addresses. Figure 1-1 illustrates the interaction of these four levels [10] .

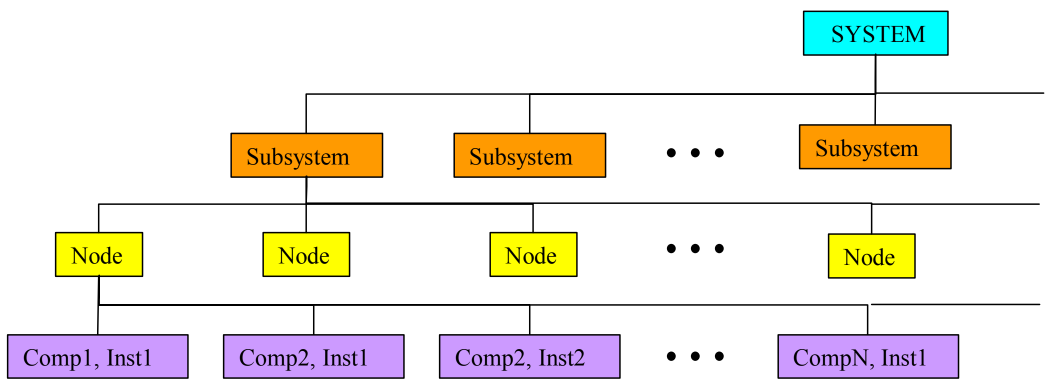

Figure 1-1: Architecture hierarchy (used from JAUS: Reference Architecture Specification, pg. 12)

1.3.3 The JAUS Component

As JAUS is a hierarchical system of components with standardized interfaces, the JAUS Component is a strictly defined entity. A distinct name and identification number defines a JAUS Component and every JAUS Component shall perform a single, cohesive function. Each Component must be able to accept and act upon the set of core JAUS command codes, as well as the input and output codes specific to the individual Component itself. A list of the core JAUS command codes may be found in Appendix A [10].

1.3.4 The JAUS Message

A JAUS message is comprised of two distinct components: the message header and the message data buffer. The message header completely defines the message's destination node, component, instance and subsystem identification, and the message's corresponding source information. The message header also consists of the JAUS command code, the number of bytes in the data buffer that the destination component can expect to receive, and information pertaining to the message properties, such as the JAUS architecture version being used. The message header data is found in Table 1-1 while the bit field layout for the message properties can be seen in Figure 1-2.

Table 1-1: Message header data format (used from JAUS: Reference Architecture Specification, pg. 54)

Field #	Field Description	Size (Bytes)
1	Message properties	2
2	Command code	2
3	Destination instance ID	1
4	Destination component ID	1
5	Destination node ID	1
6	Destination subsystem ID	1
7	Source instance ID	1
8	Source component ID	1
9	Source node ID	1
10	Source subsystem ID	1
11	Data control (bytes)	2
12	Sequence number	2
	Total Bytes	**16**

The message data buffer is composed of packed JAUS control data. Each command code has control data associated with it that is used by the system to command component behavior. In an effort to reduce the size of the JAUS messages being transmitted, and therefore reduce the required bandwidth of the system, the message data is compressed before being transmitted. To accommodate this, the JAUS code library for each component and its subsequent command codes must have the ability to pack and unpack the control data.

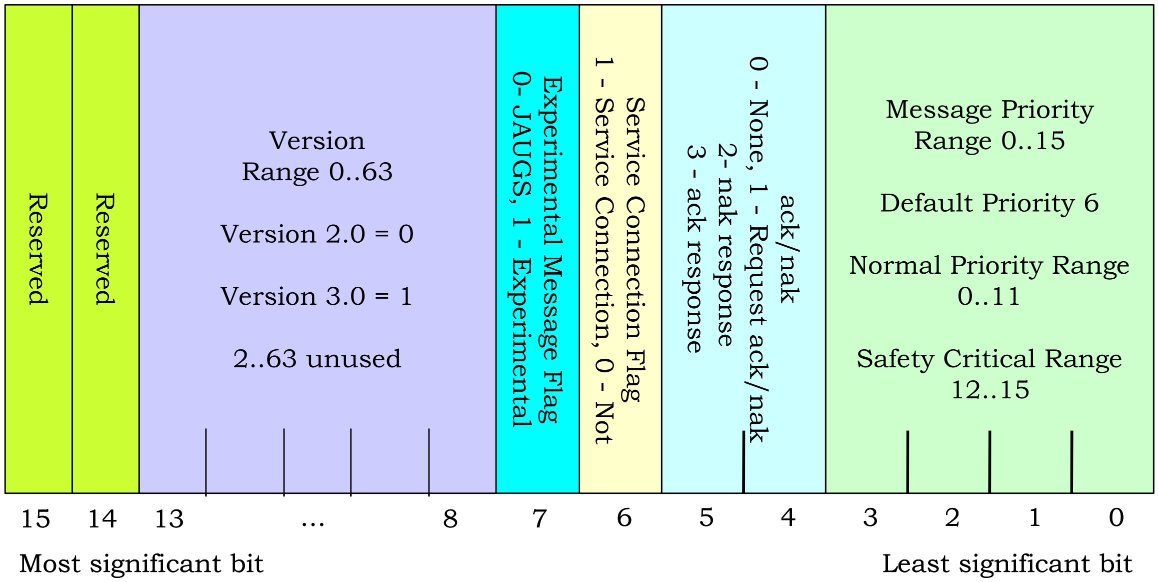

Figure 1-2: Message header properties bit layout (used from JAUS: Reference Architecture Specification, pg. 54)

CHAPTER 2
TEST PLATFORM DEVELOPMENT

To investigate the Intelligent Primitive Driver, a test platform first needed to be outfitted with a JAUS compliant control system. Due to the availability of indirect motion actuators on the system and its relatively complicated stair climbing procedures, a Remotec Andros robot was chosen for this task.

2.1 Remotec Andros Robot

Figure 2-1 depicts the Remotec Andros robot. As stated in the instruction manual, the primary design purpose of the Andros is "to provide an effective means for remote Explosive Ordnance Disposal (EOD)… Andros robots are currently used to remotely perform a variety of hazardous work tasks including explosive handling, tactical support operations, and nuclear plant maintenance" [11].

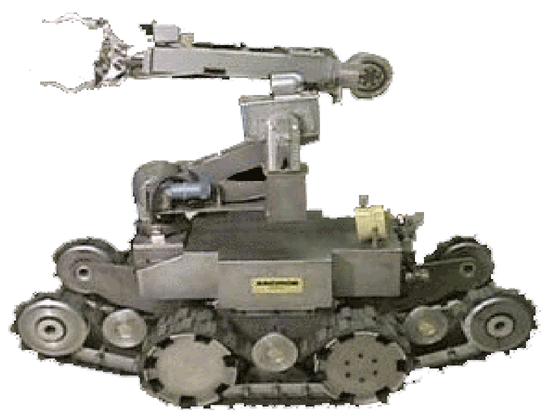

Figure 2-1: Remotec Andros robotic platform

The robot consists of a main chassis, front and rear sets of auxiliary tracks, and the main arm assembly. The tracks are made of Kevlar belts with molded internal and external urethane cleats. The auxiliary tracks are able to swing from +85 degrees to –65 degrees relative to the horizontal position. The drive motors are able to propel the Andros up forty-five degree slopes or stairs and can maintain the vehicle's position upon a slope through dynamic braking. The physical dimensions of the Andros are seen in Figure 2-2 [11].

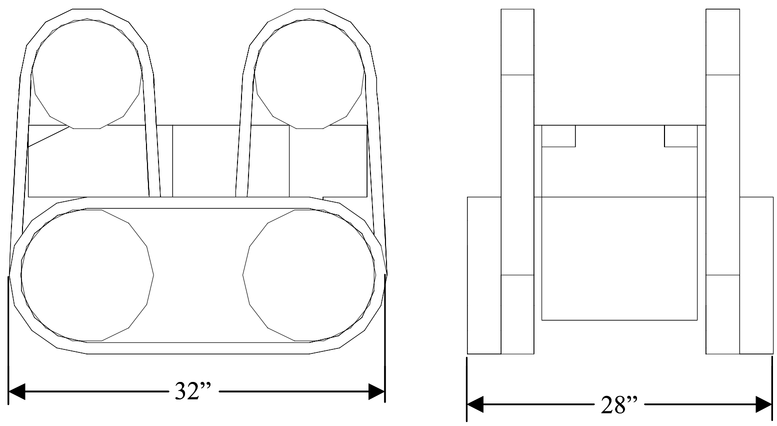

Figure 2-2: Andros physical dimensions

Figure 2-3 depicts the native Andros controller. Upon power up of the Andros system, the native control transmits an initialization string to the Andros' embedded controller and then begins the continuous output of the control data. The Andros' embedded controller will only enter the "ready" state and accept platform control data if the initialization string is properly sent and the control data is received at a maximum 5 Hz frequency. The controller uses a proprietary serial control stream to communicate with the Andros' embedded controller. The update frequency is the absolute maximum rate at which any controller may update the control data being sent to the Andros and therefore alter the Andros' intended motion.

To use the Andros robot as the test platform for the IPD system, the JAUS control system implemented on the Andros had to be completely removable from the system so that the native Andros controller could be reconnected and used to control the Andros.

To meet this interoperability constraint, the JAUS control system mimicked the serial control stream of the native controller. To achieve this, the control stream and initialization string were reversed engineered from the output of the native controller; however, the specific information pertaining to the Andros control data bytes may not be disseminated because of a nondisclosure agreement entered into between the Center for Intelligent Machines and Robotics (CIMAR) laboratory, located at the University of Florida, and Remotec, the manufacturer of the Andros platform.

Figure 2-3: Native Andros controller

2.2 Andros JAUS Controller

The Andros JAUS controller operates on a RabbitCore RCM3200 microcontroller, which utilizes a Rabbit 3000 microprocessor running at 44.2 MHz. The RCM3200 has an integrated 10/100Base-T Ethernet port and six available serial ports for communications, as well as 44 configurable, 5 volt tolerant, I/O lines. The RCM3200 runs the Dynamic C Premier Version 7.33P3 real time operating system, in which the four basic JAUS components that comprise the Andros control system were coded.

These components are the Communicator, Node Manager, Primitive Driver, and Position System. Libraries for each component were coded following the specifications found in the JAUS Reference Architecture.

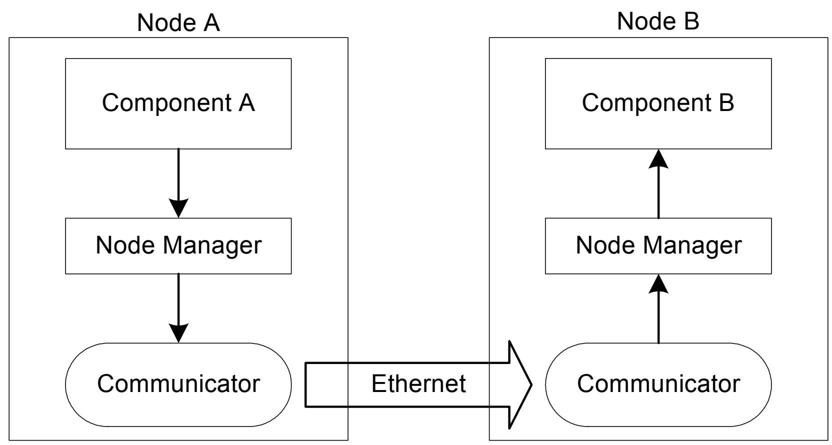

Figure 2-4: Communicator-Node Manager component interaction

Two JAUS components are directly responsible for all inter-component communications: the Communicator and the Node Manager. As such, all nodes must support both a Communicator and a Node Manager component. The Communicator allows for a single point of message entry into a subsystem while maintaining the data link between the subsystems. The Andros Communicator uses a User Datagram Protocol (UDP) Ethernet connection for all communications. The Node Manager component controls the routing of JAUS messages from one node to another. As depicted in Figure 2-4, a JAUS message is spawned within a component and then sent to the Node Manager, where the information pertaining to the destination component and destination node are attached to the message. This fully defined JAUS message is now passed on to the communicator, which handles the transmission of the message to the proper node [10].

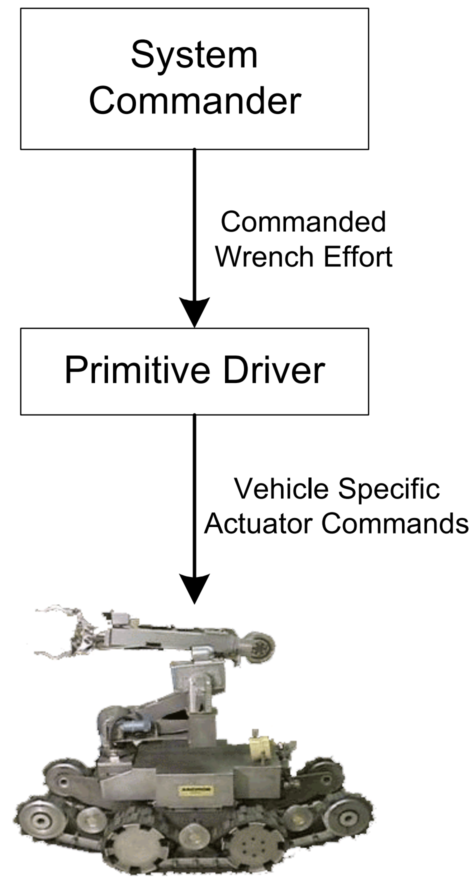

Figure 2-5: Primitive Driver component interaction

The Primitive Driver is a component, which accepts wrench commands from the System Commander and resolves them into vehicle specific actuator signals, thus initiating vehicle motion, as seen in Figure 2-5. A wrench command consists of six propulsive and six resistive elements, with each set consisting of three linear force elements and three rotational moment elements. These elements are then mapped to the three axis orthogonal coordinate system assigned to the vehicle, as shown in Figure 2-6. It is worth noting that every element of a wrench message is not necessarily applicable to every vehicle. For example, a wheeled vehicle typically only requires three elements of the wrench message to control it properly: the resistive and propulsive linear forces in the x direction and the rotational moment in the z direction [10].

The Position System is concerned solely with the determination of the vehicle's global position and orientation and is necessary for the controller to perform accurate autonomous motion. The vehicle's position and orientation information is provided to the system upon receipt of a "Query Global Pose" command from either the Operator Control Unit or the Intelligent Primitive Driver.

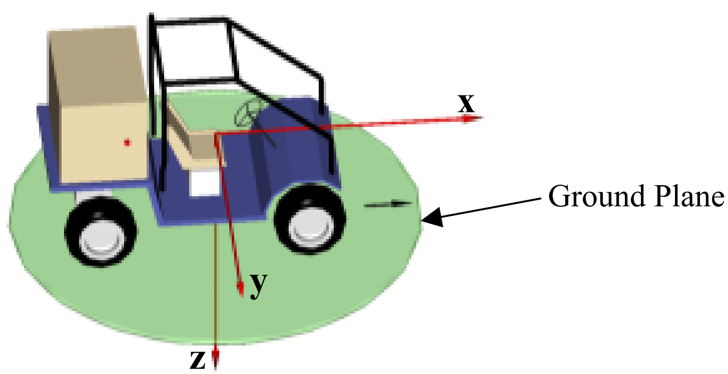

Figure 2-6: Vehicle orthogonal coordinate system

2.3 Sensor Systems

There are three main sensor systems used by the Andros system: the stair counter assembly, the auxiliary track position system, and the JAUS positioning system. These systems provide feedback essential to the controller's ability to execute behavioral commands.

2.3.1 Stair Counter Assembly

For the stair-climbing algorithms to function properly, the control system needs to know the relative position of the Andros on the stairs. Typically, a Global Positioning System (GPS) would be used to provide location information in a robotic system. However, staircases are found in or around buildings, which generally limit the effectiveness of a GPS system; if satellite transmissions to the GPS receiver are blocked by the structure housing the staircase, no position information will be acquired. Therefore, the stair counter assembly was designed to provide the relative position of the Andros upon the staircase by counting the number of stairs that the Andros has passed and is able to account for a step while moving in either direction upon a staircase. For example, while climbing the stairs, the Andros passes four steps. While attempting to move to the fifth step, a slippage error occurs and the Andros slides back down two steps. The stair counter is able to take this error into account and update the current position of the Andros upon the stairs. In this case, the Andros would now be located at the second step.

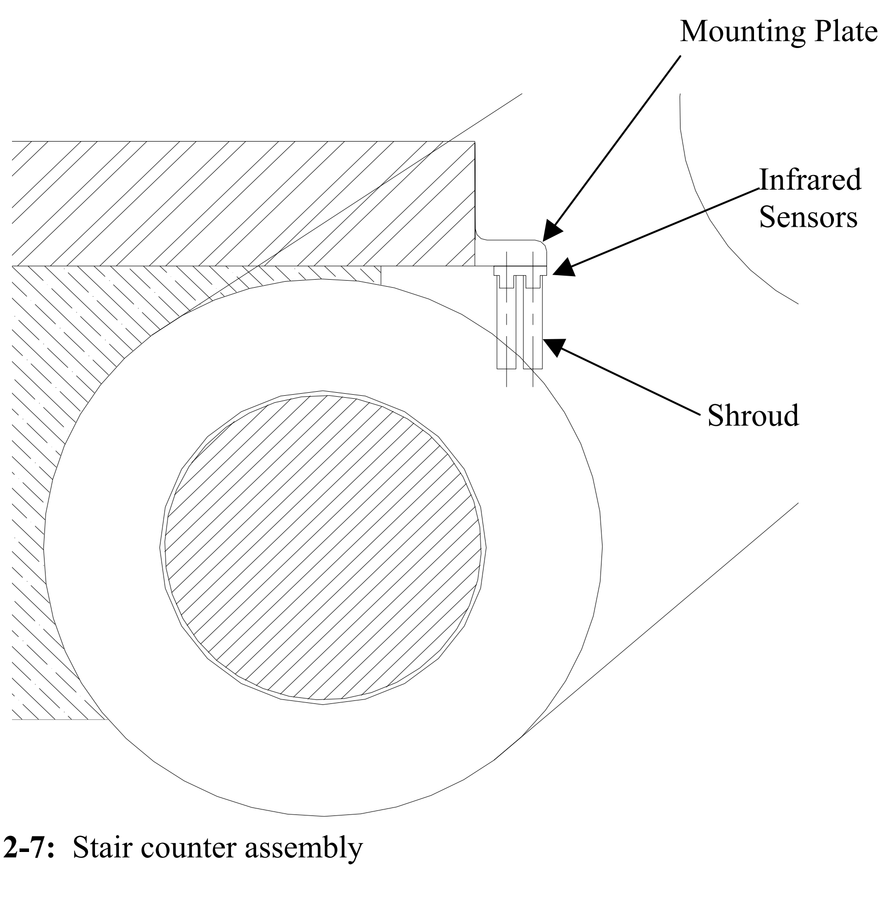

Figure 2-7: Stair counter assembly

The stair counter assembly consists of an array of two Sharp GP2D12 infrared object detectors. These sensors take continuous distance readings up to 80 cm, have an update rate of 31.25 Hz, and are interfaced by the RabbitCore microcontroller through the use of a TLC545 Texas Instruments analog-to-digital converter. The sensors are positioned one in front of the other with a one-inch offset and are orientated such that their detection beams are perpendicular to the long axis of the Andros, as seen in Figure 2-7; the sensors point directly at the plane upon which the Andros is situated. Both sensors are shrouded to limit the amount of interference they may receive from ambient light sources and from each other.

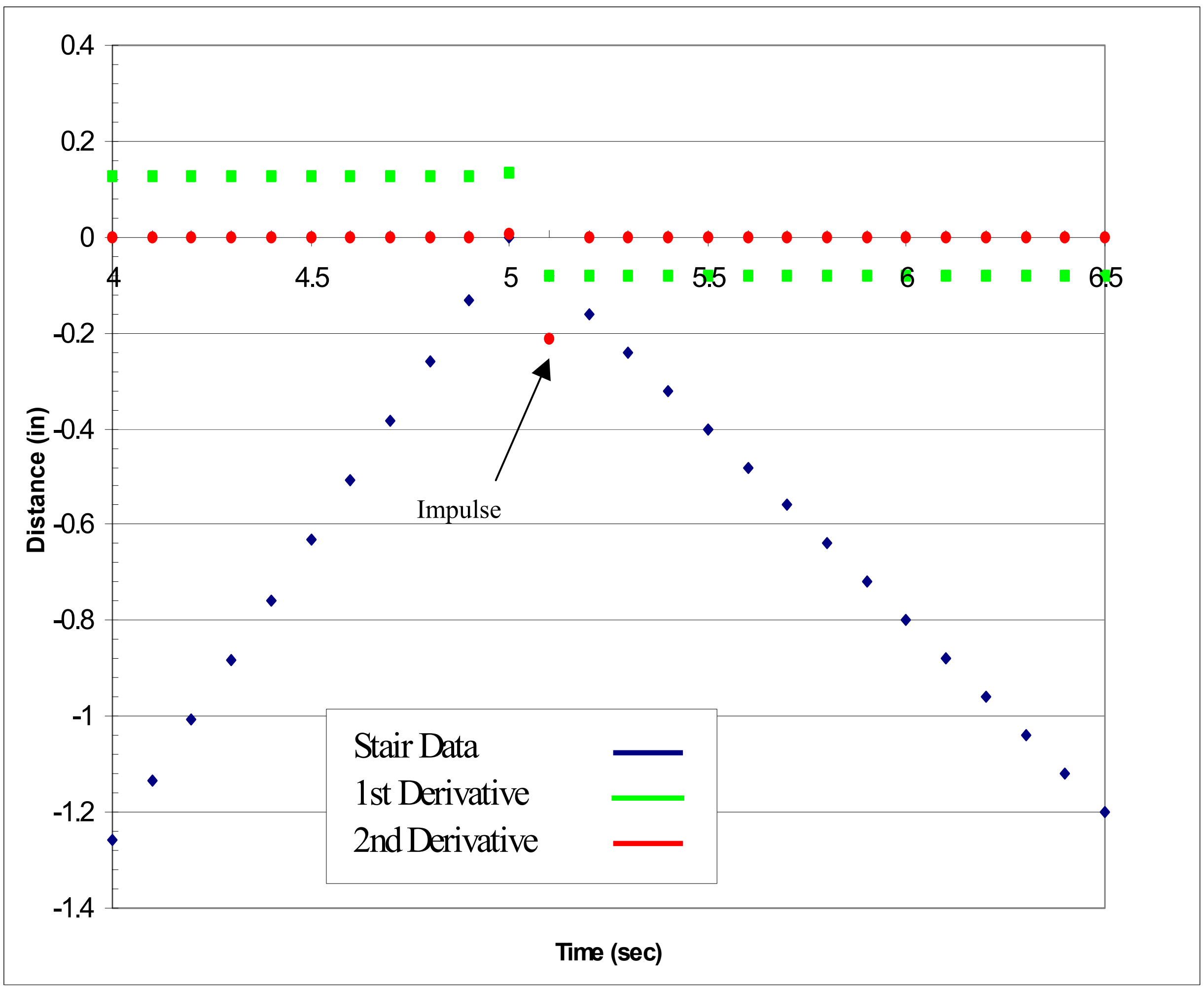

Figure 2-8: Infrared sensor stair representation

Each infrared sensor constructs a digital representation of the staircase as the Andros moves on it by determining the distance from the sensor to the staircase. By assuming that the vehicle maintains a relatively constant velocity upon the staircase, the derivative of the stair data can be found by simply taking the difference of the current data point and the previous data point collected. The second derivative of the stair data can now be obtained by taking the difference of the first derivative data. Figure 2-8 depicts a sample stair data representation for a single infrared detector and the corresponding first and second derivative data. The graph has been cropped to emphasize the range of pertinent data. The data for the stair representation was simulated using a mathematical model of the sensor system, with the first and second derivative data obtained as outlined above. For the simulation, the rise of a single step was set at 8 inches, the run was set at 10 inches, and a 0.1 second sampling rate was used. Derivation of the mathematical model of the sensor system can be found in Appendix B.

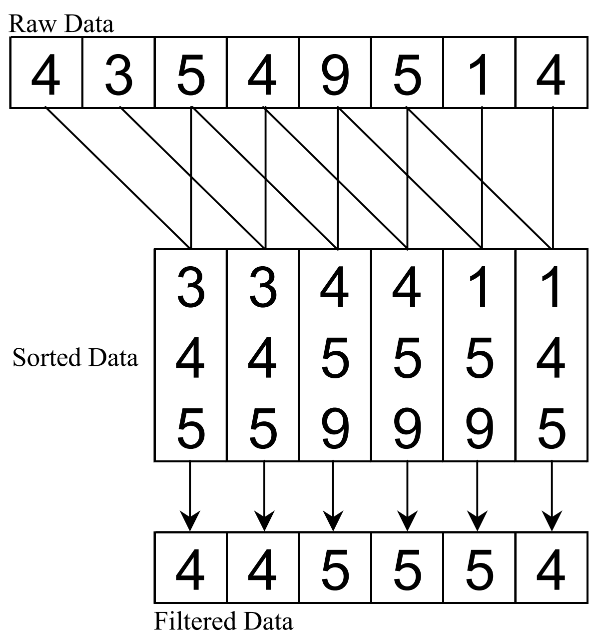

Figure 2-9: Median filter example

To account for noise in the infrared detection signals, the first derivative data is filtered using a three element median filter before the second derivative is calculated. Median filtering is a non-linear filtering technique useful for the suppression of impulse noise and the smoothing of edges. It works by taking a set number of data points and determining the mathematical median of them, which then replaces the data point. An example of a three element median filter is shown in Figure 2-9. In this example, three elements of the raw data are grouped together and sorted from low to high. In this configuration, the median of the data is simply the middle of the three numbers, which is now the filtered data point [12].

The impulses in the second derivative data represent inflection points on the stairs; the interior concave stair corner will yield a positive impulse while an exterior convex stair corner will yield a negative impulse. The system accounts for the stairs that the Andros has passed by detecting these impulses; if the impulse that is less than a given threshold value, a stair is counted. The direction of travel upon the stairs, and therefore whether to add or subtract a stair to the total number counted, is determined by which sensor detected the impulse first. If the front detector senses the impulse before the rear, then a stair is added. If the rear detector senses the impulse before the front, then a stair is subtracted from the total.

The computation of the first and second derivatives of the stair representation and the application of the median filter induces a delay in the detection of the stair inflections. Each of the derivative computations account for a one-cycle delay, while the median filter accounts for a three-cycle delay for a total delay of five cycles. As the detectors are polled at a rate of 20 Hz, the total delay time is 0.25 seconds between acquisition of the stair data and detection of a stair inflection.

The stair counter assembly also has an “end of world” capability, meaning that it can be used for detection of drop-offs and ledges. The system uses this function to determine the edge of a landing when descending a set of stairs. The edge is detected by monitoring the output of the infrared sensors: the distances measured will remain constant until the sensor reaches a point when it is directly over the first step. At this point, there will be a drastic change in the measured distance, which the system interprets as the edge of the landing and the beginning of the stairs.

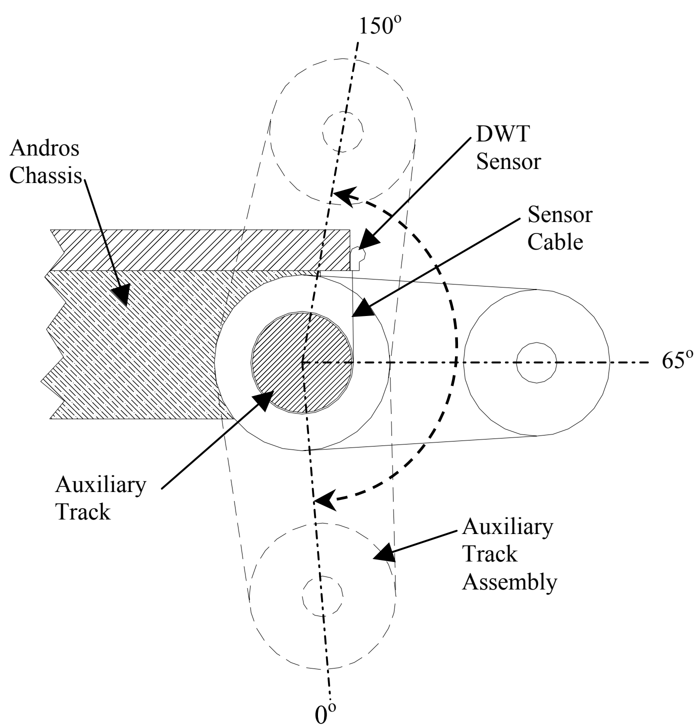

Figure 2-10: Auxiliary track position system setup

2.3.2 Auxiliary Track Positioning System

A Draw-Wire Transducer (DWT) sensor, manufactured by UniMeasure, is used to measure the angle of the Andros auxiliary tracks, one each for the front and rear tracks.

This sensor was used over other forms of angular sensors, such as rotary or optical encoders, because of its ease of setup and robust mechanical interface; there are no gears to become jammed with dirt or optical sensors to become inoperable because of grime. The DWT is simply a spring-loaded potentiometer connected to a three-foot long cable. The DWT sensor is attached to the chassis of the Andros, with the draw cable being wrapped around the auxiliary track drum once and then attached to it via a 10-32 cap-head screw. The setup is shown in Figure 2-10.

A 5-volt reference voltage is applied to the sensor, which will return from 0 to 5 volts proportional to the length of cable drawn from the sensor. The governing equation for the angle measured by the system is given in Equation 2-1, where V_{high} and V_{low} are the upper and lower voltage bounds observed by the sensor and $V_{measured}$ is the sensor voltage.

$$\theta = \left[\frac{\left(V_{measured} - V_{low}\right)}{\left(V_{high} - V_{low}\right)} \right] * 150^\circ \qquad \textbf{(2 – 1)}$$

The DWT sensors are interfaced by the RabbitCore microcontroller through the use of a TLC545 Texas Instruments analog-to-digital converter.

2.3.3 JAUS Positioning System

As there is no GPS being used on the Andros system, the position system is comprised solely of a PNI Corporation TCM2-20 tilt-compensated module to update the Andros heading. The compass heading of the Andros is filtered using a three element median filter, as previously described.

The TCM2-20 is based upon PNI Corp.'s proprietary triaxial magnetometer system and its biaxial electrolytic inclinometer. When level, it is accurate to within +/- 0.5

degrees RMS with 0.1-degree resolution. This changes to +/- 1 degree RMS when the system is on an incline. It communicates via a 38.6 kBaud, RS-232 serial connection and is set to run at a sampling rate of 20 Hz.

The digital compass experiences a computation delay, much in the same manner as the stair counter system. The three element median filtering produces a delay of three cycles, which corresponds to a delay of 0.15 seconds between acquisition of the heading data and realization of the filtered heading.

2.4 Operator Control Unit

In order for the Andros' JAUS control system to be fully operable, an Operator Control Unit (OCU) was developed on a Gateway Solo 3350 laptop with the Linux Redhat 8.0 operating system. A Netgear ME102 Wireless Access Point is used to establish a wireless Ethernet connection with the Andros JAUS controller. The access point uses the IEEE 802.11b standard and transmits up to 11 Mpbs at 2.4 to 2.5 GHz.

To be able to communicate with the JAUS components located on the Andros platform, the OCU needed to have information pertinent to the construction of the JAUS messages required to command these components available. Therefore, JAUS message libraries for the Primitive Driver and Position System components were coded for the Linux operating system. These libraries were adapted from code obtained from Dr. Jeff Wit of Wintech, Inc. Communicator and Node Manager components were also coded for the Linux operating system to allow the OCU to interact with the Andros' JAUS components. All of the OCU components were developed in parallel with the corresponding components used on the Rabbit microcontroller.

The OCU console was developed using the Linux Curses library. This was chosen for the basis of the user interface because it allows for the easy manipulation of the terminal display regardless of the terminal type used. The Curses library only supports the use of single byte characters for display, but this was more than adequate to meet the needs of this OCU.

CHAPTER 3
INTELLIGENT PRIMITIVE DRIVER DEFINITION AND APPLICATION

This chapter defines the Intelligent Primitive Driver (IPD) in terms of the JAUS architecture. It then details the application of the Intelligent Primitive Driver to the Andros system.

3.1 Component Definition

For the IPD to be considered a JAUS component, it needs to fit into the rigid framework that comprises the architecture. Therefore, the component will be defined as per the specifications detailed in the JAUS Reference Architecture: the component function, input and output messages, and description will be defined. The component has been assigned the component identification number of 99 [10].

3.1.1 Component Function

The Intelligent Primitive Driver component is responsible for the control of all indirect motion related actuators. An indirect motion related actuator is any actuator that affects the ability of a platform to perform a commanded motion, but does not perform that motion itself. For example, on the Andros robot, the front and rear auxiliary tracks are considered indirect motion actuators. They are responsible for aiding the Andros in moving over rough and uneven terrain, but do not directly produce the motion.

3.1.2 Component I/O

The IPD will accept any of the JAUS core input and output messages, as well as the "Set Wrench Effort" and "Query Global Pose" commands. These commands are as defined in Appendix A. Table 3-1 shows the user defined set of input and output messages and their corresponding unique command codes. The input and output command sets will be completely defined in the "Input and Output Commands" section later in this chapter.

Table 3-1: User defined input and output JAUS commands

	Command Code	Description
IN	0x1FF1	Set Auxiliary Actuator
IN	0x1FF2 - 0x1FF9	Set Behavior X
IN	0x3FF1	Query Auxiliary Actuator
OUT	0x3FF2 - 0x3FF9	Query Behavior X
OUT	0x5FF1	Report Auxiliary Actuators
OUT	0x5FF2 - 0x5FF9	Report Behavior X

3.1.3 Component Description

The IPD defines a command interface to control any available indirect motion actuators available upon a platform and any subsequent set of vehicle behaviors that would utilize these actuators. Further, the IPD is able to issue wrench commands to a primitive driver component. Automation of vehicle behaviors is accomplished to ease operator burden when the JAUS system is acting in tele-operational mode and to decrease high-level processor load when the system is acting in full-autonomous mode. Similar to the Primitive Driver component, the IPD does not imply any specific platform in its definition and therefore is not completely defined until it has been applied to the control system of a particular platform.

Figure 3-1 illustrates the most basic implementation of the Intelligent Primitive Driver into the JAUS system architecture. The System Commander may send commanded wrench efforts to the primitive driver to initiate vehicle motion or any of the aforementioned IPD command codes to the Intelligent Primitive Driver either to control an indirect motion actuator or start a vehicle behavior. If a command is sent to begin a vehicle behavior, the IPD is able to send vehicle specific commands directly to the vehicle to control the indirect motion actuators. Also, if vehicle motion is necessary to accomplish the behavior, the IPD may send commanded wrench efforts to the primitive driver component to incite vehicle motion.

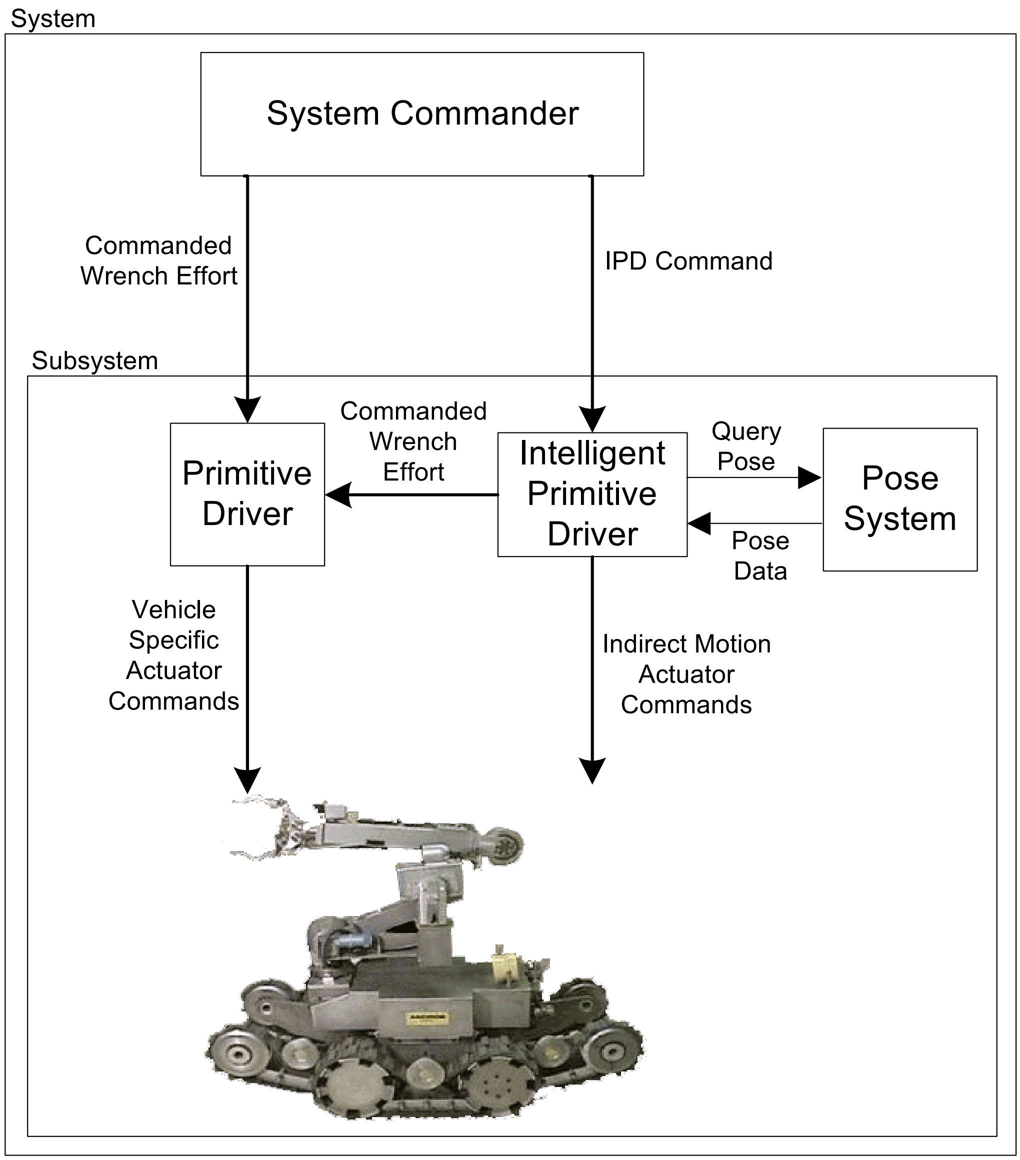

Figure 3-1: Intelligent Primitive Driver implementation

3.1.4 IPD Input and Output Messages

This section will define the set of user-defined input and output messages used by the IPD.

3.1.4.1 "Set Auxiliary Actuators"

The "Set Auxiliary Actuators" message controls the indirect motion actuators available on the platform. The command consists of four available linear actuator fields and four available rotational actuator fields. Each field in the command may be mapped directly to a single actuator of corresponding type, either linear or rotational, for control. The interpretation of the numerical limits of the data fields allow for a wide array of actuator control. The limits may be interpreted as a percentage of the maximum speed to move the actuator at, a percentage of the maximum distance to move the actuator to, or even interpreted as a blind forward/reverse/stop control message. The data fields and vector-mapping table is for this command is found in Table 3-2.

Table 3-2: "Set Auxiliary Actuators" command fields

Field #	Name	Type	Units	Interpretation
1	Presence Vector	Unsigned Short	N/A	See Mapping Table that Follows
2	Aux Actuator 1	Short Integer	Percent	Scaled Integer Lower Limit: -100 Upper Limit: 100
3	Aux Actuator 2			
4	Aux Actuator 3			
5	Aux Actuator 4			
6	Aux Actuator 5	Short Integer	Radians	Scaled Integer Lower Limit: $-\pi$ Upper Limit: π
7	Aux Actuator 6			
8	Aux Actuator 7			
9	Aux Actuator 8			

Vector to Data Field Mapping for Presence Vector of Above Command

Vector Bit	7	6	5	4	3	2	1	0
Data Field	9	8	7	6	5	4	3	2

3.1.4.2 "Query Auxiliary Actuators"

The "Query Auxiliary Actuators" message will cause the IPD to reply to the requesting component with a "Report Auxiliary Actuators" message. The command field is shown in Table 3-3.

Table 3-3: "Query Auxiliary Actuators" command field

Field #	Name	Type	Units	Interpretation
1	Presence Vector	Unsigned Short	N/A	See "Report Auxiliary Actuator" Message

3.1.4.3 "Report Auxiliary Actuators"

The "Report Auxiliary Actuators" command message provides the receiving component with the current values of the commanded "Set Auxiliary Actuators" message. The message data and mapping of the presence vector for the "Report Auxiliary Actuators" message are the same as the "Set Auxiliary Actuators" message, as seen in Table 3-2.

3.1.4.4 "Set Behavior X"

The "Set Behavior X" message is used to initiate the behavioral control algorithms, which are to be applied to the individual platform. The message has been issued the command code range of 0xFFF2 to 0xFFF9 to allow for 8 vehicle specific behavior codes to be applied to the control system. The corresponding behavior number, from 1 to 8, replaces the 'X' in the command code names in the code definition. The data fields are delineated into 4 unsigned integers and 4 full integers, giving the design engineer a wide range of available variables to transmit any necessary behavioral data to the system, such as latitude and longitude, heading, or distance to travel. As such, a specific behavior may be defined by all, some, or none of the available data fields. The data fields and vector-mapping table is for this command is found in Table 3-4.

Table 3-4: "Set Behavior X" command fields

Field #	Name	Type	Units	Interpretation
1	Presence Vector	Unsigned Short	N/A	See Mapping Table that Follows
2	Behavioral Parameter 1			
3	Behavioral Parameter 2	Unsigned Int	N/A	User defined fields
4	Behavioral Parameter 3			
5	Behavioral Parameter 4			
6	Behavioral Parameter 5			
7	Behavioral Parameter 6	Float	N/A	User defined fields
8	Behavioral Parameter 7			
9	Behavioral Parameter 8			

Vector to Data Field Mapping for Presence Vector of Above Command

Vector Bit	7	6	5	4	3	2	1	0
Data Field	9	8	7	6	5	4	3	2

3.1.4.5 "Query Behavior X"

The "Query Behavior X" message will cause the IPD to reply to the requesting component with a "Report Behavior X" message. The command field is shown in Table 3-5.

Table 3-5: "Query Behavior X" command field

Field #	Name	Type	Units	Interpretation
1	Presence Vector	Unsigned Short	N/A	See "Report Behavior X" Message

3.1.4.6 "Report Behavior X"

The "Report Behavior X" command message provides the receiving component with the current behavioral data being performed by the platform. The message data and mapping of the presence vector for the “Report Behavior X” messages are the same as the “Set Behavior X” message, as seen in Table 3-4.

3.2 Intelligent Primitive Driver Application

An Intelligent Primitive Driver component must be applied to a vehicle to fully define the functionality of the behavioral and auxiliary actuator controls as no vehicle specific data is given in the JAUS definition of the component. This section will discuss the four main areas that comprise the application of the IPD to the Remotec Andros test platform. These areas include: a discussion of the basis of the intelligence used in the control algorithms, the functionality of the component as it pertains to the Andros robot, an explanation of the program logic used in the controller, and an explanation of the stair motion algorithms used by the controller.

3.2.1 Fuzzy Logic

Professor Lotfi Zadeh at the University of California in Berkley developed fuzzy Logic in 1965 as a method of processing data by allowing partial set membership rather than classical precise set membership or non-membership. Fuzzy Logic is based upon a human intelligence model; Professor Zadeh reasoned that people do not need precise, numerical information as input and yet they are capable of highly adaptive control. As intuitive as this approach of control may appear, it was not applied to an actual control system until the 1970’s, mainly due to inadequacies in small-computer capabilities at the time [13].

Fuzzy Logic provides a simple way to arrive at a definite conclusion based upon vague, ambiguous, noisy or imprecise information. The logic model focuses on what a system should do rather than trying to understand how the system works and concentrates on the problem rather than attempting to represent the system mathematically, which may be impossible, especially in the case of nonlinear systems. Fuzzy Logic incorporates a simple, rule-based "if X and Y then Z" problem solving control approach. This approach relies upon an empirically based model, which is dependent on the design engineer's control experience. The system is inherently robust as it does not require precise inputs, can be designed to fail safely if an input is lost, and despite the possible wide variation of the input signal, the output is always a smooth control signal. As the design engineer defines the rules that govern the system, the controller can easily be modified to improve or dramatically alter system performance. Finally, any sensor that presents the controller with an indication of the system's actions, regardless of sensor cost or precision, is a viable candidate for the fuzzy controller [13].

The fuzzy logic control model was applied to the four major intelligence functions of the control system: the stair protocol arbiter, the platform alignment controller, the auxiliary track controller, and the task used to realign the Andros upon the stairs after a slippage error has occurred. Each of these functions will be discussed as they are encountered in the descriptions of the control algorithms to follow.

3.2.2 Functionality

To utilize the control capabilities of the Intelligent Primitive Driver, the command messages associated with the component must be completely defined to interact with the platform. The "Set Auxiliary Actuators" command must be mapped to control the platform's indirect motion actuators. Vehicle behaviors must be assigned a control number corresponding to one of the available "Set Behavior" commands, the requisite control data for the behavior must be defined and assigned to the proper command variable and the behavioral algorithms themselves must be characterized and coded for use.

As shown in Table 3-2, the "Set Auxiliary Actuators" command message has eight assignable auxiliary actuators command fields. Following this convention, the front auxiliary track of the Andros robot is denoted as "Aux Actuator 5" and the rear auxiliary track is denoted as "Aux Actuator 6." These fields were chosen as the auxiliary tracks of the Andros may be moved to a specific angle through the use of the Auxiliary Track Position System, described in Chapter 2.3.

The behavioral controls of the IPD are used to control the motion of the Andros on a set of stairs, both ascending and descending. The ascending motion behavior has been mapped to "Set Behavior 1" with the descending motion mapped to "Set Behavior 2." Both behaviors use the same the control data variables, which have been assigned to the data fields shown in Table 3-6.

Table 3-6: "Set Behavior" 1 and 2 control data

Field #	Name	Description
2	Behavioral Parameter 1	Rise of a Single Step (in.)
3	Behavioral Parameter 2	Run of a Single Step (in.)
4	Behavioral Parameter 3	Total Number of Steps
5	Behavioral Parameter 4	Not Used
6	Behavioral Parameter 5	Stair Heading (degrees)
7	Behavioral Parameter 6	Not Used
8	Behavioral Parameter 7	Not Used
9	Behavioral Parameter 8	Not Used

3.2.3 Program Logic

Figure 3-2 depicts the logic flow diagram of the Intelligent Primitive Driver upon startup. This diagram is only representative of the Intelligent Primitive Driver and not the JAUS system on the whole. The receipt and execution of JAUS commands by standard components have been omitted because, for these instances, the controller behaves in a manner consistent with a standard JAUS controller. Upon startup of the Andros controller, the IPD enters a "ready" state and awaits the reception of an IPD command message. The receipt of one of these commands causes the IPD to assume control of the Andros and execute the desired commanded function.

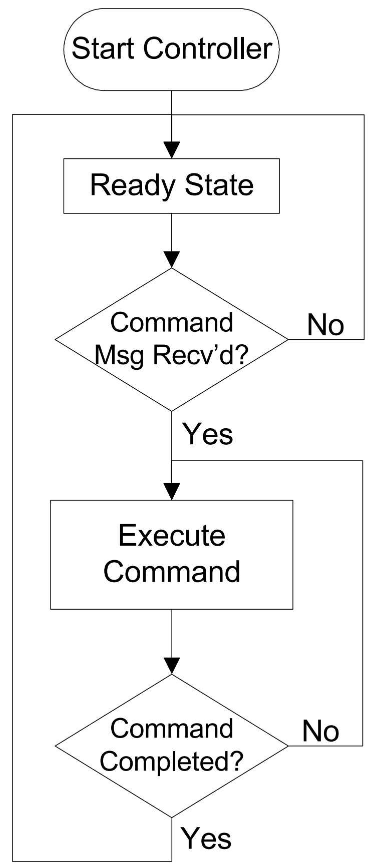

Figure 3-2: Intelligent Primitive Driver logic flow diagram

Upon receipt of a "Set Auxiliary Actuators" command, the IPD moves into the algorithm as shown in Figure 3-3. The IPD uses the Auxiliary Track Positioning System, as outlined in Chapter 2.3, to determine the current angle of the auxiliary track. Once the current angle has been determined, the algorithm checks to see in which of three possible fuzzy sets the current track angle lies: less than the desired angle, greater than the desired angle, or around the desired angle. The IPD will determine the direction, if any, to rotate the track dependent upon which set the current track angle lies. If the current track angle is less than the desired track angle, the track is rotated counterclockwise. If the current track angle is greater than the desired track angle, the track is rotated clockwise. If the track angle is equal to the desired track angle, within an angular tolerance of +/-1 degree, the track is not rotated and the controller IPD returns to the ready state. The two-degree total tolerance was determined heuristically based upon the maximum update rate of the controller, the precision of the Auxiliary Track Positioning System and the overall speed of motion of the auxiliary tracks while rotating. This algorithm is followed independent of which auxiliary track is commanded to move.

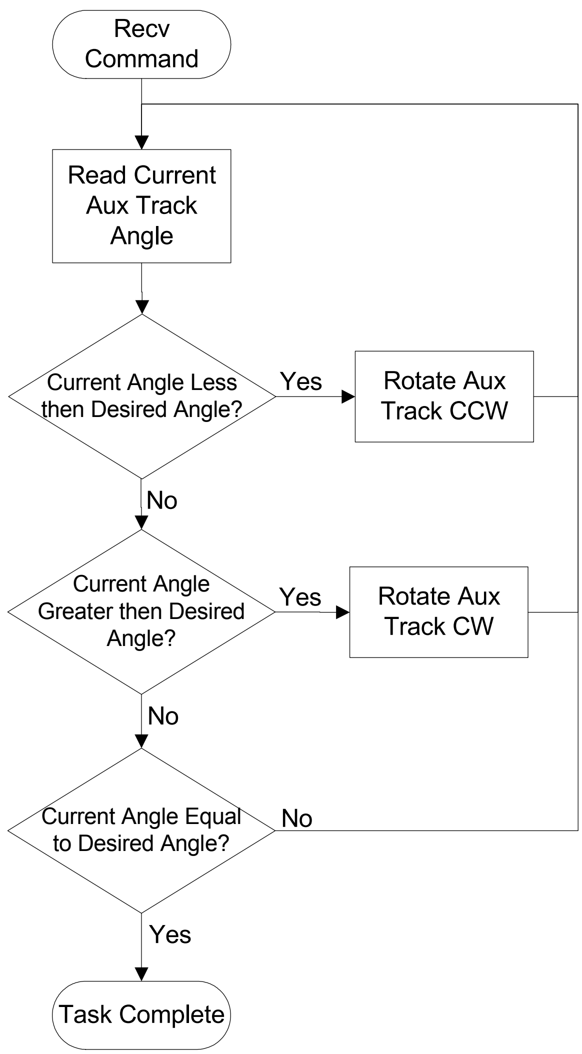

Figure 3-3: “Set Auxiliary Actuator” algorithm

Upon receipt of a "Set Behavior" command, the IPD executes the algorithm shown in Figure 3-4. The IPD calculates the total dimensions of the stairs from the control data received along with the command, as seen in Table 3-6. Through simple geometric equations, the IPD determines the total length, **L**, and pitch angle, β, of the stairs, as seen in Figure 3-5. Next, the IPD aligns the heading of the Andros platform with the heading of the staircase.

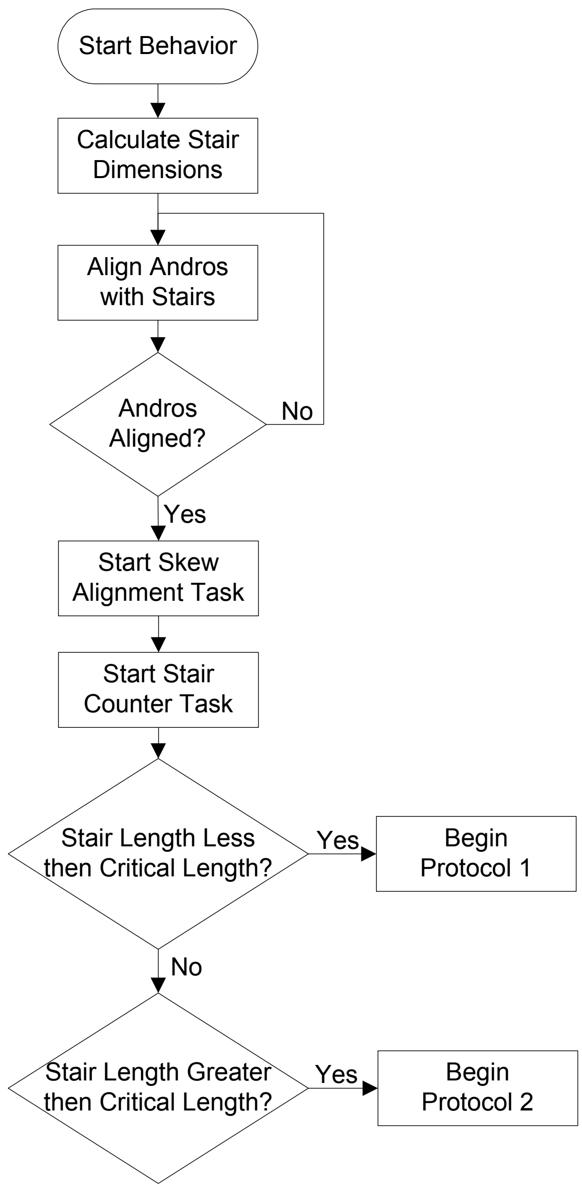

Figure 3-4: Opening sequence of "Set Behavior X" algorithm

The align platform function is another of the fuzzy logic functions. It calculates the difference between the reported stair heading and the current heading of the Andros, obtained from the Andros' onboard digital compass. The absolute value of the difference is then compared against the defined fuzzy sets.

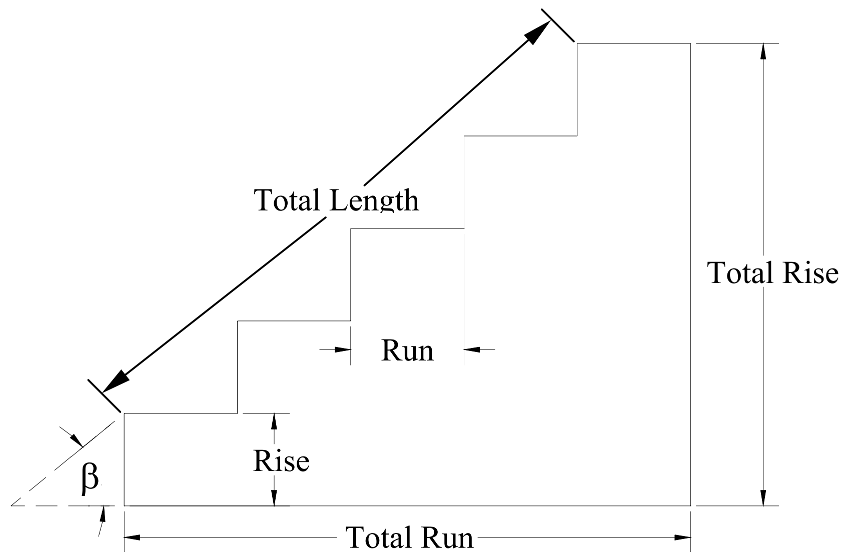

Figure 3-5: Stair data representation

If the absolute value is greater than 90 degrees, the Andros is rotated at its maximum speed. If the absolute value is between 90 and 60 degrees of error, the Andros is rotated at 75% of its maximum speed. If the absolute value is between 60 and 25 degree of error, the Andros is rotated at 50% of its maximum speed. Finally, if the absolute value is less than 25 degrees, the Andros is rotated at 25% of its maximum speed. This function has an allowable angular tolerance of +/-4 degrees. The direction of the commanded rotation is also dependent upon a fuzzy interpretation of the angular difference. If the absolute value of the difference is less than 180 degrees and the actual difference is negative, the Andros is rotated counterclockwise at the desired speed. If the absolute value of the difference is less than 180 degrees and the actual difference is positive, the Andros is rotated clockwise at the desired speed. The direction of rotation relative to the actual value of the difference is reversed if the absolute value of the difference is found to be greater than 180 degrees. This ensures that the Andros takes the shortest route possible to reach the commanded heading.

Once the Andros has been aligned, the IPD starts the skew alignment task. This is a separate program thread, which runs parallel to the main stair control thread. The purpose of the skew alignment task is to detect and counter any angular slippage of the Andros upon the staircase. Slip detection is accomplished by monitoring the difference between the current heading of the Andros and the heading of the stairs, similar to the align platform function.

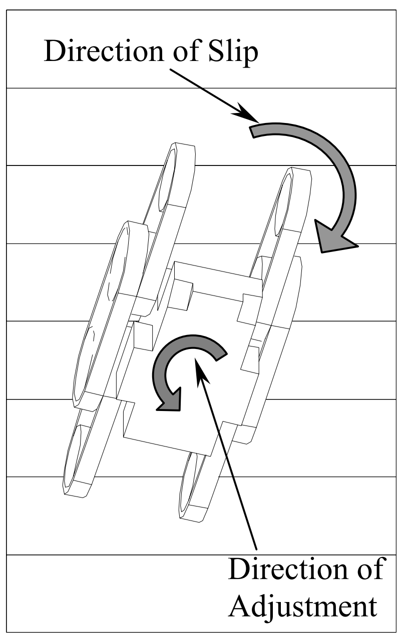

Figure 3-6: Skew alignment task correction

If an angular slippage error is detected, the skew alignment task counters it by rotating the Andros in the direction opposite of the slippage. (Figure 3-6) This, again, is a fuzzy logic task as the slippage error is grouped into broad sets of possible angular error. If the error is greater than 5 degrees but less than 10, the Andros is rotated at 15%

of its maximum speed. If the error is between 10 and 25 degrees, the Andros is rotated at 25% of its maximum speed. At greater than 25 degrees, the error begins to approach a non-recoverable state, and the Andros is stopped so that appropriate measures can be taken to safely move it off of the staircase. Once the heading error is less than the 6 degrees of angular tolerance, the Andros rotation is halted. The linear motion control command of the Andros is never affected by the skew alignment task, except for the case in which the error is greater than 25 degrees, as the Andros' embedded controller allows for both linear and rotational motion components to be commanded at the same time.

After the skew alignment task has been spawned, the IPD begins the stair counter task. The stair counter task uses the stair counter assembly, as described in chapter 2.3, to count the number of steps the Andros has successfully passed and also to account for any possible linear slippage error. The number of steps must be counted to provide the IPD with the relative position of the Andros upon the staircase to trigger stair motion events, such as auxiliary track motions.

The next step for the IPD to complete is to determine which stair protocol should be evoked. This is accomplished by determining if the total length, **L**, of the staircase is less than or greater than the critical length. The critical length is defined as 1.5 times the total length of the Andros. If the total length is less than the critical length, protocol 1 is chosen. If the total length is greater than the critical length, protocol 2 is chosen. The reason for the demarcation between the two protocols is simple: a staircase with a total length equal to or less than the critical length simply does not have enough available length to adequately carry out the full range of stair climbing motions. Also, the majority of the stair motions are meant to ensure that the Andros is stable upon the staircase and that the tracks do not lose traction. This is not a concern on short staircases as it is on longer ones.

It should be noted that the portion of the stair motion algorithm, as seen in Figure 3-4, is the same for both the ascend and the descend behavior commands. Also independent of the direction of travel upon the stairs is the protocol arbiter. Protocol 1 is designated for control of the Andros on a set of stairs less than the critical length in both the ascend and descend commands, protocol 2 corresponds to control of the Andros on a set of stairs longer than the critical length in both the ascend and descend commands.

Once the stair protocol has been carried out, the controller reverts back to its ready state and awaits further commands from the System Commander.

3.2.4 Ascend Protocol 1

This stair motion protocol deals with the situation in which the total length of the staircase is less than the critical length. First, the front auxiliary tracks of the Andros are aligned with the angle of the stairs, β. (Figure 3-7a) Then, the Andros is simply started up the staircase. The protocol concludes as the Andros passes the last step of the staircase and is safely situated on the top landing, as seen in Figure 3-7c.

3.2.5 Ascend Protocol 2

The second ascend stair motion protocol deals with the situation where the total length of the staircase is longer than the critical length. In this situation, the full range of control steps are necessary to ensure that the Andros makes it safely to the top of the staircase. The first stage of the process is to align the front auxiliary tracks of the Andros with the angle of the stairs, β, and the rear auxiliary tracks parallel to the ground. (Figure 3-7a) This stage occurs while the Andros is preparing to climb the staircase, therefore there are no steps associated with it. In the next stage, the Andros is started up the staircase. After two steps have passed, the front auxiliary tracks are aligned parallel with the line of the stairs, as seen in Figure 3-7b. The Andros continues on in this third stage alignment until the last step of the staircase is reached. As this step is reached, the last stage is initiated. The front auxiliary tracks of the Andros are moved downward to catch the Andros as it passes the apex of the stairs. This can be seen in Figure 3-7c.

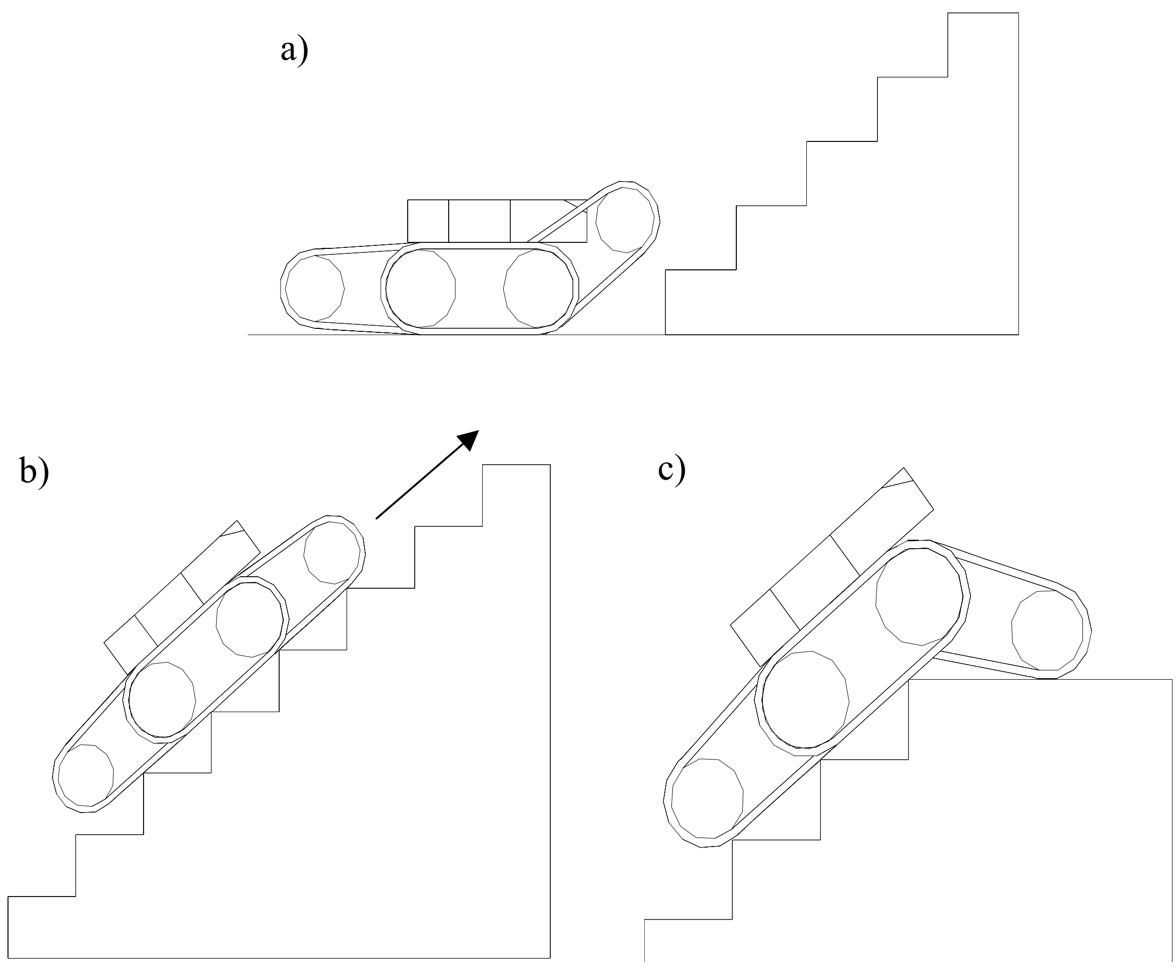

Figure 3-7: Ascend stair motion protocol: a) Align front auxiliary tracks with stair angle; b) Andros moving on stairs; c) Andros moving over apex of stairs

3.2.6 Descend Protocol 1

As with ascend protocol 1, descend protocol 1 is evoked when the total length of the staircase to be traversed is less than the critical length. The Andros is moved to the edge of the staircase and the front auxiliary tracks are moved downward to catch the Andros as it crosses the apex of the stairs, as seen in Figure 3-8a. Next, the Andros is simply driven over the edge of the steps and the front auxiliary tracks are moved parallel to the ground plane. Once the Andros has reached the floor level, it is moved away from the staircase.

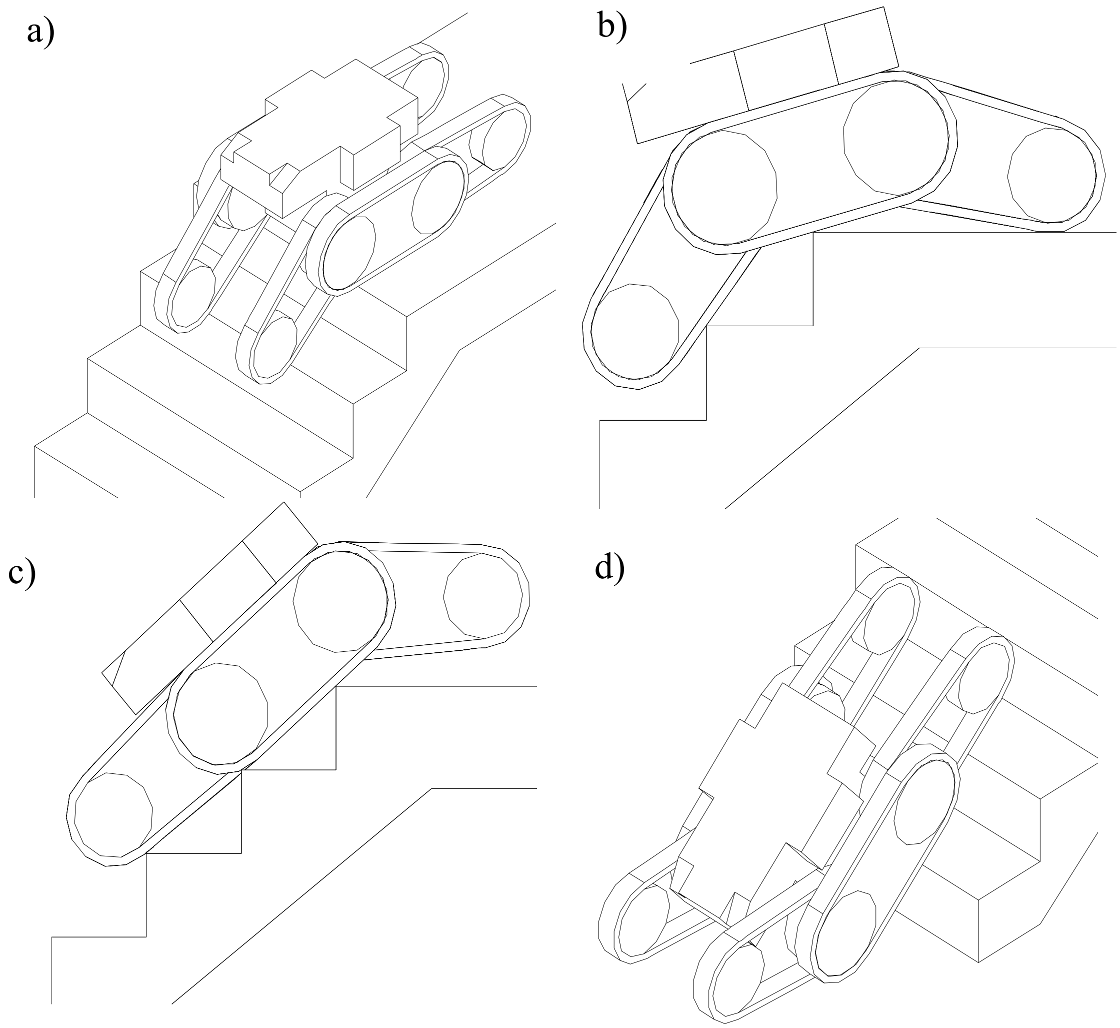

Figure 3-8: Descend stair motion protocol: a) front auxiliary track moved to catch Andros; b) rear auxiliary track moved to push Andros over apex; c) align front auxiliary track with stairs; d) front auxiliary tracks moved to catch Andros at stair landing

3.2.7 Descend Protocol 2

Descend protocol 2 occurs when the total length of the staircase to be traversed down is longer than the critical length. The Andros is moved to the edge of the stairs and the front auxiliary tracks are moved downward to catch the Andros as it crosses the apex of the stairs. (Figure 3-8a) While this is occurring, the rear auxiliary tracks are moved downward to help rotate the Andros down to the plane of the staircase. (Figure 3-8b) The Andros is moved forward until the main tracks come in contact with the stairs, as seen in Figure 3-8c. The controller uses the “end of world” capability of the stair counter assembly to determine the edge of the stairs, as discussed in Chapter 2.3. The Andros is moved further forward and rear auxiliary tracks are moved to a horizontal position. The Andros is moved down the steps until the third to last step is reached. Then, the front auxiliary tracks are moved to parallel to the ground plane, which allows the Andros to make an easy transition to the plane of the floor. (Figure 3-8d) The Andros is then moved off and away from the staircase.

CHAPTER 4
TESTING AND RESULTS

This chapter describes the testing of the individual subsystems associated with the Andros' Intelligent Primitive Driver (IPD). Testing procedures and results will be reported for the stair-counter assembly, the auxiliary track positioning system and the heading alignment controller.

4.1 Stair-Counter Assembly

4.1.1 Testing Procedure

The stair counter assembly was tested to ascertain whether or not the concept would accurately count the number of stairs the platform has passed and therefore, be a viable addition to the positioning system. The first step to accomplish this task was to calibrate the infrared sensors that the system uses. This was done to correlate the values returned from the analog-to-digital converter (ADC), which are used to interface the Sharp GP2D12 infrared object detectors and the Rabbit RCM3200 microcontroller, to their corresponding distances. To do this, a target was first placed a known distance away from the infrared sensor. Then, one hundred readings of the value returned from the infrared sensor through the ADC were recorded and their average obtained. The target was then moved to another known distance and the average value was again found. This process was repeated for distances measuring 4, 6, 8, 10, 12, 14 inches. The data points were plotted using Excel and several regression curves were applied to determine which type best fit the data. It was determined that the non-linear power function shown in

Figure 4-1 best described the output. The equation shown was then used in the code to convert the ADC readings to distances for the testing.

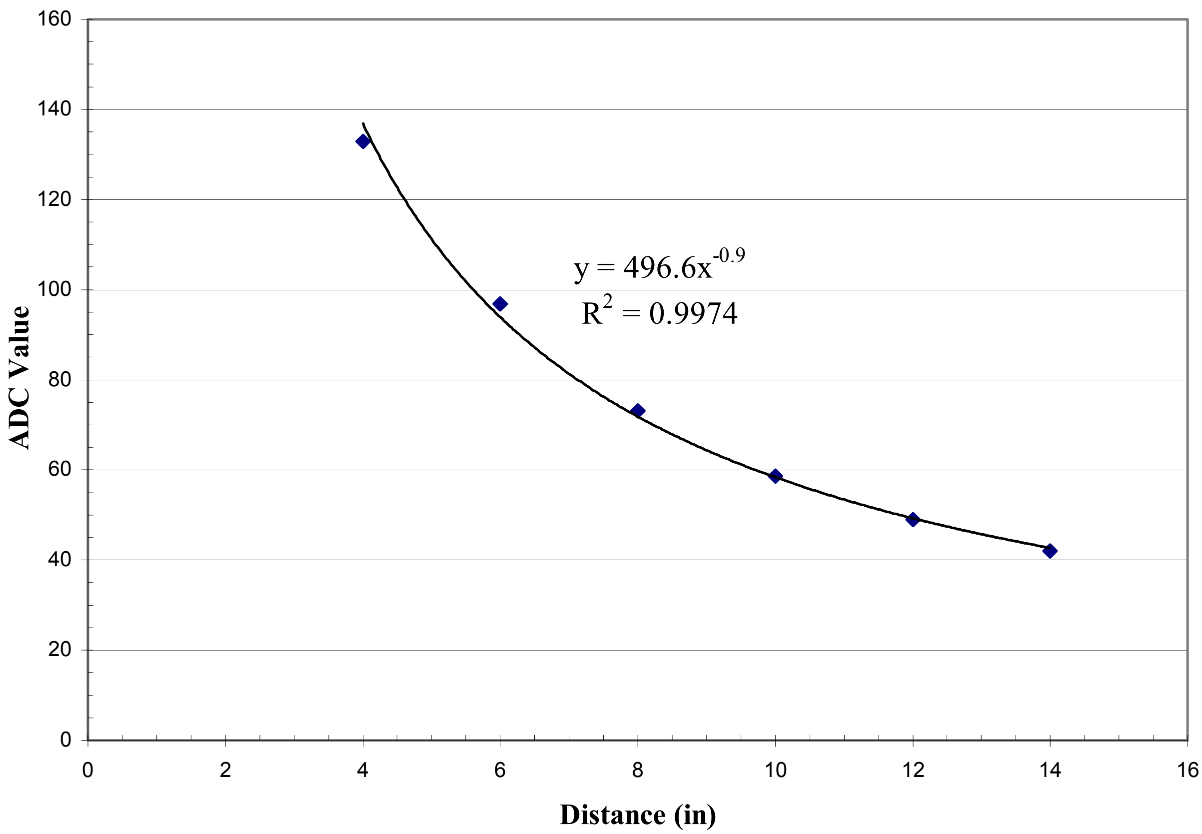

Figure 4-1: Calibration data for Sharp GP2D12 infrared object detectors

Once the infrared sensor had been calibrated, a method of gathering the test data from the staircase needed to be devised. To accomplish this, a testing frame made from 80/20 extruded aluminum framing system was constructed. The 80/20 was chosen for this task as it was easily assembled and the extruded grooves were ideal for constraining the motion of the test sled along the major axis of the stairs. The test sled housed the Rabbit microcontroller as well as one of the Sharp GP2D12 infrared object detectors to acquire the range data. The sled was then tethered to a 12-volt gear motor to pull it along at a relatively constant velocity. The entire assembly was placed upon the stairs, as seen in Figure 4-2.

Figure 4-2: Stair-counter testing assembly

The system recorded a reading of the range data from the surface of the staircase to the infrared sensor once every 50 milliseconds. This delay was chosen, as it is longer than the 32-millisecond update rate of the infrared sensors and therefore ensured that no duplicate readings were accidentally recorded by the system. The microcontroller then determined the first derivative of the data, applied a three element median filter to it, and then determined the second derivative. A Dell Inspiron 8100 laptop computer then recorded this data set. Communication between the Dell and the Rabbit were made via the 57.6-kBaud connection provided for diagnostic purposes by the microcontroller. The test was run a total of ten times on two separate staircases.

The stair-counter testing also had a secondary purpose: to empirically determine a second derivative threshold value for the stair-counter control code. The stair-counter code uses the second derivative values to determine if a stair has been passed and should therefore be counted. However, detecting a single second derivative value for this purpose is futile; a range of second derivative values that correspond to a stair edge may be encountered. The system may also experience subtle negative second derivative peaks due to uneven motion of the Andros upon the stairs. To account for all of this, the second derivative value is compared to a threshold value. If the value is less than the threshold, it is determined to be a stair edge and is counted.

4.1.2 Results

The data collected from the stair-counter testing rig was placed into an Excel spreadsheet for graphing purposes. The data was analyzed to determine when the second derivative of the stair representation occurred with respect to the recorded inflection point, or outside edge, of the stairs.

Figure 4-3a illustrates the first set of sample data from the acquired from the testing. The dark blue line represents the detected outline of the staircase, while the red line represents the calculated second derivative of this data. Clearly visible are the two large negative impulses, each one corresponding to one of the stair peaks shown. As designed, each impulse is logged 0.25 seconds after the stair peak is experienced. The same is true of Figures 4-3b and 4-3c; each one has two large, negative impulses that occur 0.25 seconds after their corresponding stair peaks are detected. Also, in each of these figures, the negative impulses are less than –0.2, while none of the tertiary negative impulses approach this value. Therefore, this was determined to be the optimum threshold value for the system to use.

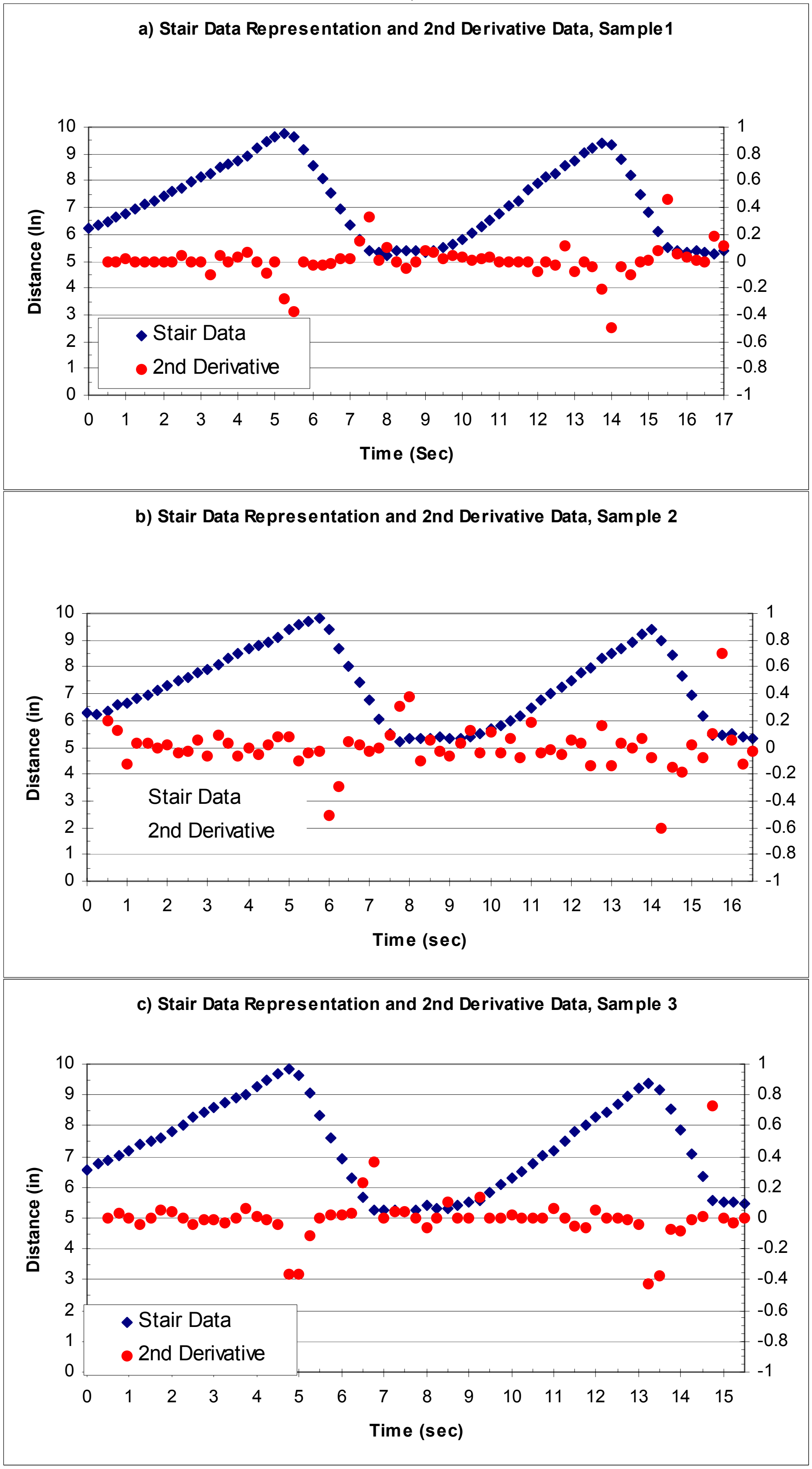

Figure 4-3: Stair-counter data: a) sample 1, b) sample 2, c) sample 3

4.2 Auxiliary Track Positioning System

4.2.1 Testing Procedure

The setup for the auxiliary track positioning system testing had two steps. First, the maximum and minimum values returned to the system by the Draw Wire Transducer (DWT) were obtained. These values are used in conjunction with Equation 2-1 to convert the measured DWT value to an angular measurement. The maximum value occurs when the auxiliary track has been rotated to its topmost point, while the minimum value occurs when the auxiliary track has been rotated to its bottommost point; these points correspond to 150 degrees and 0 degrees, respectively. At each of these two points, one hundred values of the DWT were taken and then an average value was obtained.

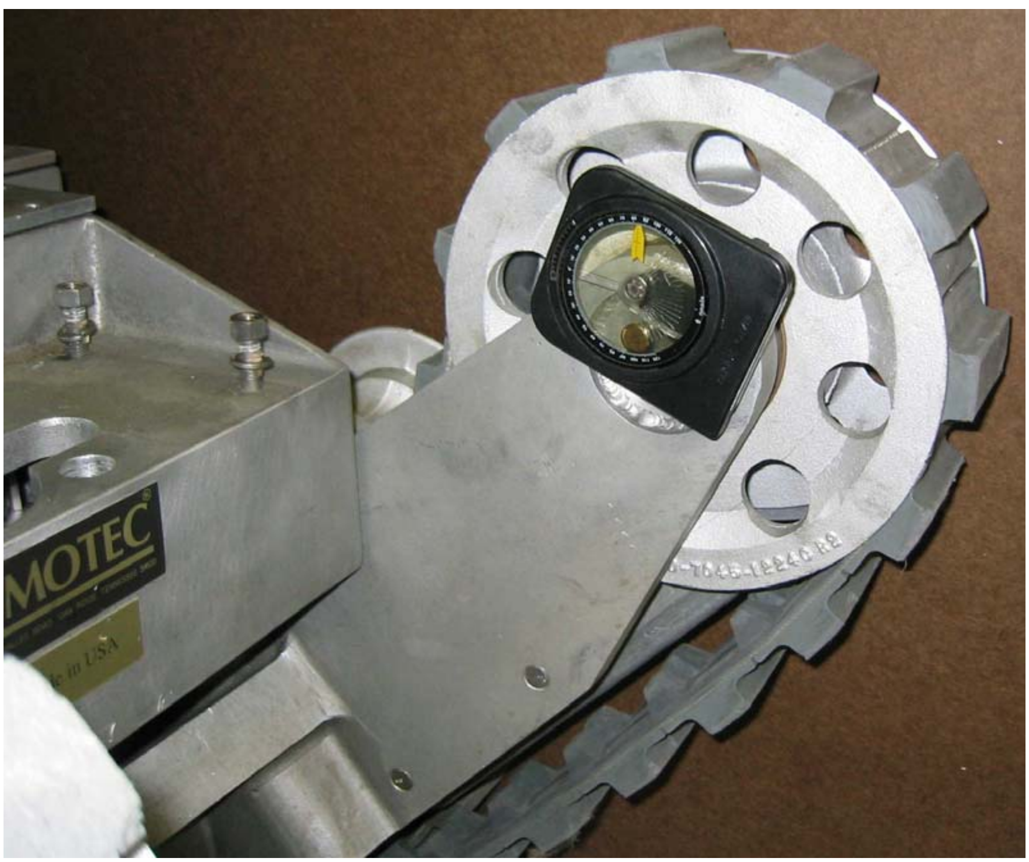

Figure 4-4: Plumb compass attached to Andros auxiliary track

The second step in the setup was to affix a plumb compass to the auxiliary track so that an independent angle measurement of the position of the track could be made. A plumb compass is comprised of a needle, weighted at the bottom, along with an adjustable faceplate. By attaching the body of the plumb compass to a structure, the relative angle of the structure with respect to ground may be found when the structure is rotated. The Andros auxiliary tracks were rotated to their bottommost point and the adjustable face of the plumb compass was then adjusted such that the compass needle pointed to 0 degrees. In this configuration, the compass needle pointed to 150 degrees when the auxiliary tracks were rotated to their topmost position, which corresponds to the measurement convention utilized by the controller. Figure 4-4 illustrates the plumb compass attached to the Andros' auxiliary track.

To test the commands for the auxiliary track positioning system, the Operator Control Unit (OCU) was used to send the experimental JAUS message, "Set Auxiliary Actuators," to the Andros' JAUS controller, along with a commanded angular position of the auxiliary track. The initial set of commanded angles were used to determine the minimum allowable tolerance that could be used by the system without the system becoming unstable. In this case, the system would be unstable if the controller continually reversed the direction of the auxiliary track in an attempt to move the track to the desired angle and be within the given tolerance. The minimum allowable tolerance for the system was found to be +/- 1 degree. Following this, another set of "Set Auxiliary Actuators" commands were sent to the system and the commanded angle, the angle returned by the DWT sensor, and the angle read off of the plumb compass were recorded.

4.2.2 Results

Figure 4-5 shows a plot of the controller commanded halt angle and the actuator halt angle versus the commanded angle sent to the controller from the OCU. The controller commanded halt angle is the angle returned to the controller from the DWT sensor at the instant that the controller issued the halt rotation command. The actuator halt angle is the angle read from the plumb compass and corresponds to the angle of the auxiliary track after the rotation has actually stopped. Both sets of data points have a linear distribution across the entirety of the available range of motion, as illustrated by the applied trend lines and their respective R^2 values. The controller commanded halt angle and the actuator halt angle are identical at 0 degrees and 150 degrees because there are physical stops at these points of rotation on the Andros frame.

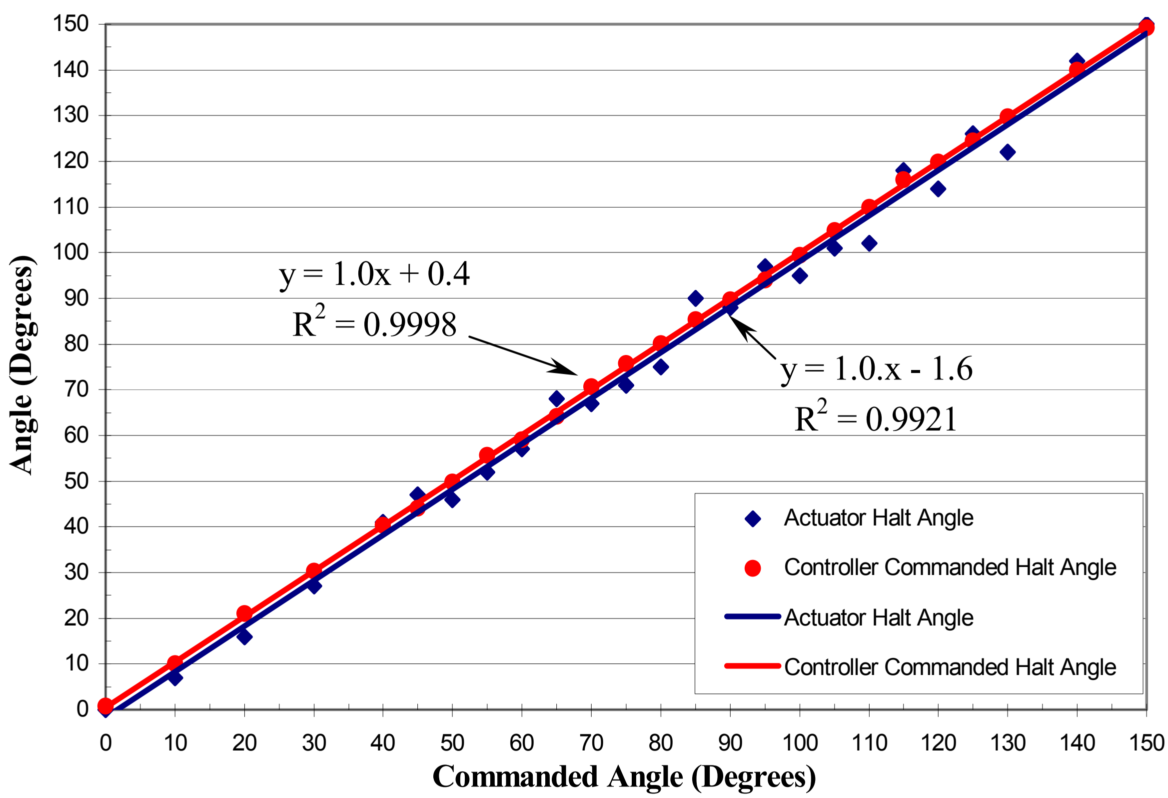

Figure 4-5: Controller commanded halt angle and actuator halt angle versus operator commanded angle

Figure 4-6 depicts a plot of the error between the commanded angle sent from the OCU and the controller commanded halt error and the actuator halt angle. The controller commanded halt error is the error between the commanded angle and the auxiliary track angle at the instant that the controller issued the halt command and is recorded from the DWT sensor. The actuator halt error is the error between the commanded angle and the auxiliary track angle after the track actually stops and is recorded from the plumb compass. The average error value returned from the DWT sensor is 0.5 degrees with a standard deviation of 0.3, whereas the average error value returned from the plumb compass is 3.4 degrees with a standard deviation of 2.2.

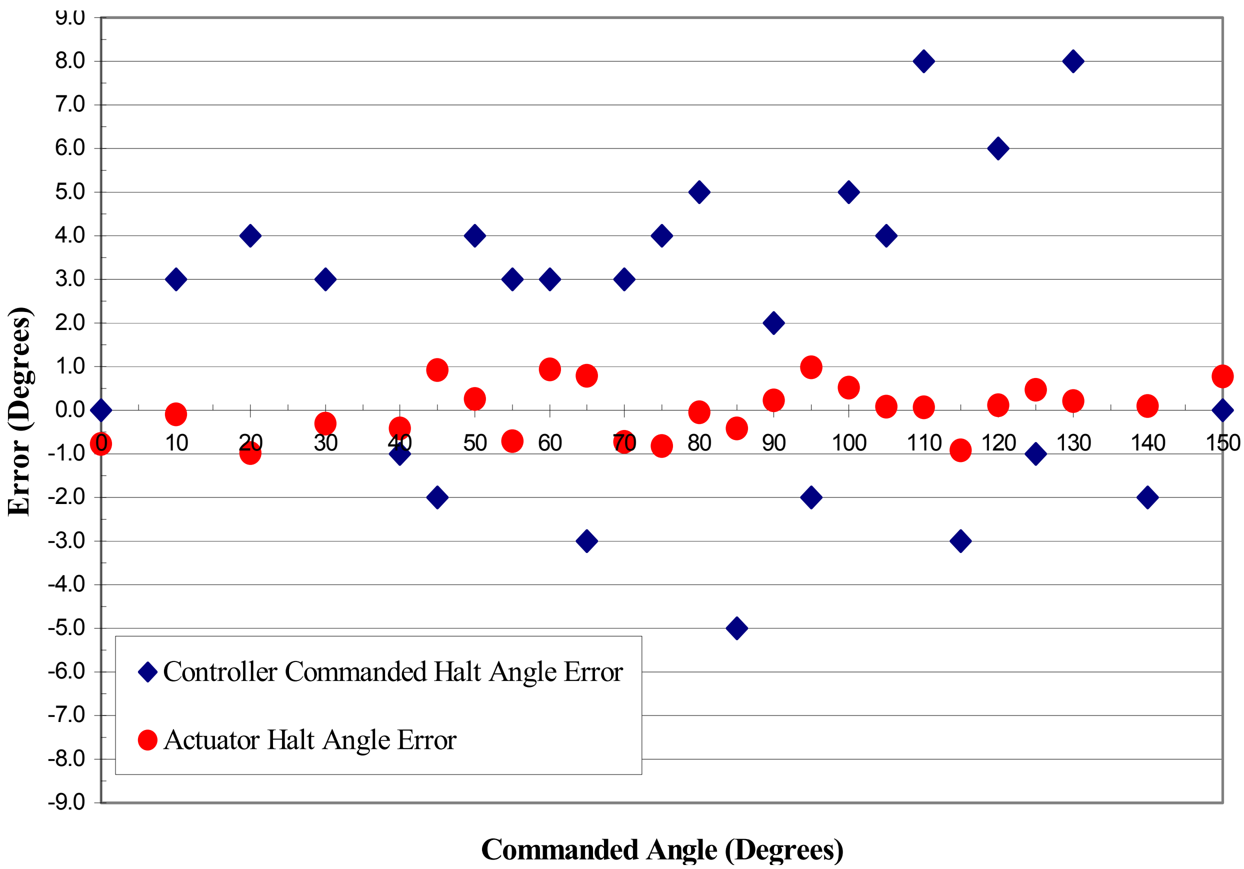

Figure 4-6: Controller commanded halt error and actuator halt error versus operator commanded angle

4.3 Heading Alignment Controller

4.3.1 Testing Procedure

The first step necessary to test the heading alignment controller was to calibrate the PNI Corporation TCM2-20 digital compass. A serial connection was established between the digital compass and a Dell Inspiron 8100 laptop computer and the digital compass was placed into polling mode. Then, the compass was given the command to start its calibration cycle and the Andros was slowly rotated in a circle for approximately four minutes. Once this was completed, a series of "Set Behavior 1" commands were sent from the OCU to the Andros JAUS controller, along with the desired heading. The JAUS controller was coded such that the "Set Behavior 1" command initiated the heading alignment code. The first series of tests were administered to determine the minimum allowable tolerance value that the system could accept without the system becoming unstable. In this situation, the system would be unstable if the controller continually reversed the rotation of the Andros in an attempt to move the Andros to the desired heading. From these initial tests, the allowable tolerance for the system was found to be +/- 4 degrees. After this was completed, another set of "Set Behavior 1" commands were sent from the OCU and the commanded heading, compass heading, and final Andros heading were recorded. The compass heading is the value that the digital compass last reported to the controller to illicit a stop rotation command. The final Andros heading is the compass measure of the heading of the Andros when the rotation actually stops.

4.3.2 Results

Figure 4-7 depicts a plot of the controller commanded halt heading and the vehicle halt heading versus the desired heading sent from the OCU. The controller commanded halt heading is the heading returned to the controller at the instant that the controller issues a stop vehicle rotation command. The vehicle halt heading is the heading of the vehicle when the vehicle actually stopped its rotation. Both sets of data have a linear distribution across the entire range of rotation, as illustrated by the applied trend lines and their respective R^2 values.

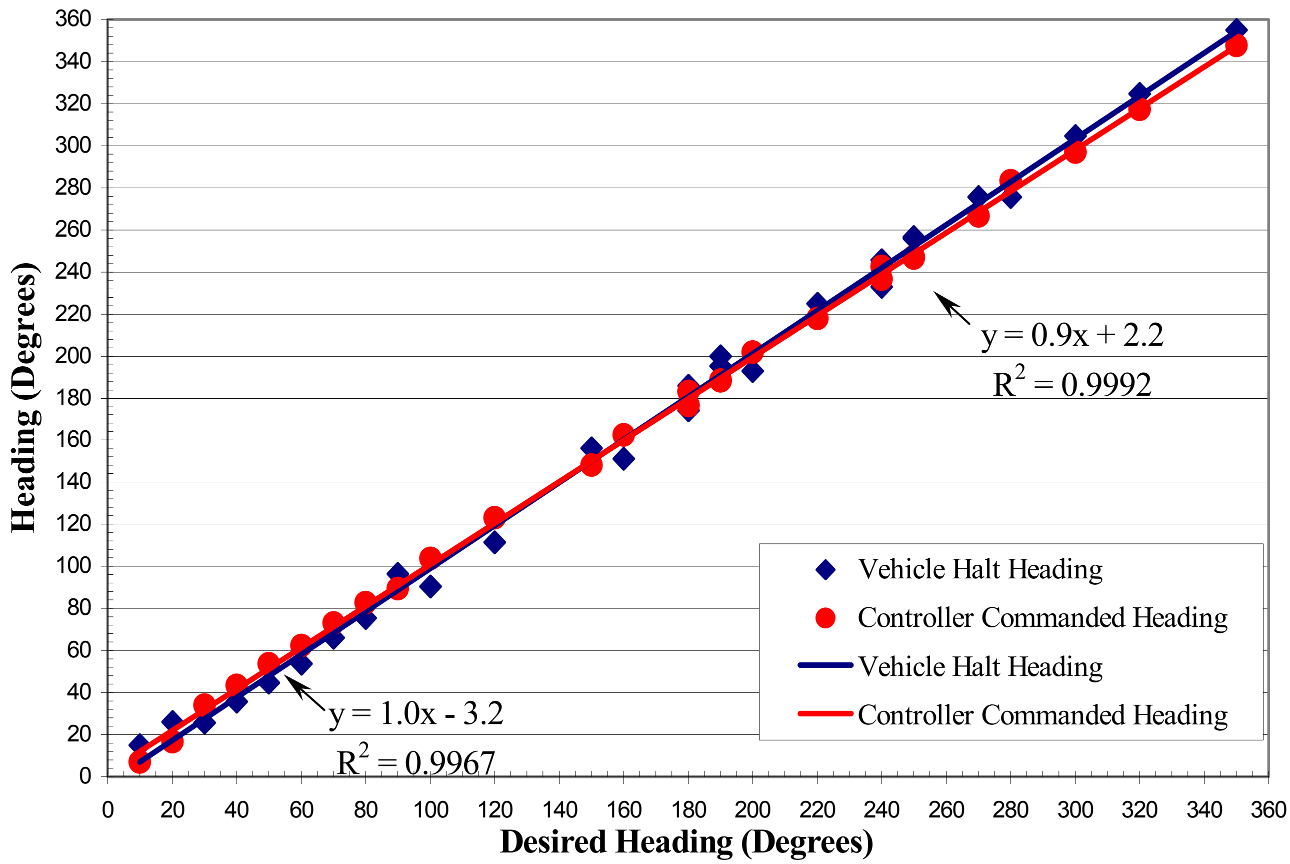

Figure 4-7: Controller commanded halt heading and vehicle halt heading versus operator commanded heading

Figure 4-8 depicts a plot of the controller commanded heading error and the vehicle heading error versus the desired heading sent from the OCU. The controller commanded heading error is the error between the desired heading and the heading of the vehicle when the controller issues a stop rotation command. The vehicle heading error is the error between the desired heading and the heading of the vehicle when its rotation is actually stopped. The average error value for the controller heading is 2.8 degrees with a standard deviation of 0.8, whereas the average error value for the final Andros heading is 5.9 degrees with a standard deviation of 1.7.

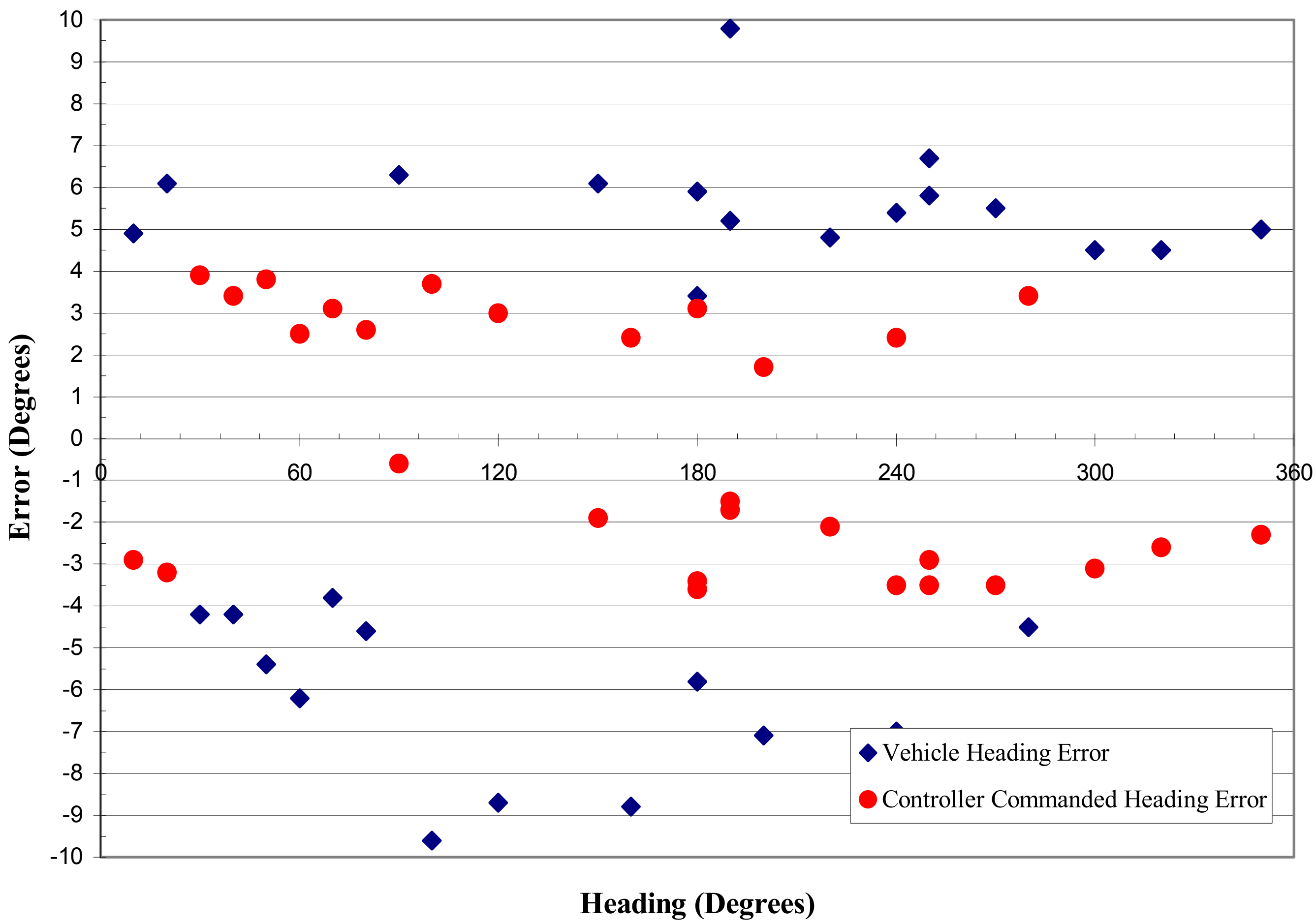

Figure 4-8: Controller commanded heading error and vehicle heading error versus operator commanded heading

CHAPTER 5
CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

This research focused on the development of the Intelligent Primitive Driver (IPD), an experimental, low-level, intelligence component for the Joint Architecture for Unmanned Systems (JAUS). Three objectives were necessary to accomplish this task. First, the IPD had to meet the criteria of a fully defined, standard JAUS component. Second, a testing platform had to made JAUS compliant and the functionality of the IPD had to be applied to the testing platform. Finally, the intelligence components of the IPD that were applied to the testing platform had to be tested.

A standard JAUS component is defined through four criteria: the function of the component, the accepted input and output messages that the component may handle, and the full description of the component. In these terms, the IPD was fully constrained to the JAUS architecture. The purpose was defined as being responsible for the control of all indirect motion related actuators. A full set of input and output messages were defined to meet the JAUS message standards. The component description defined the implementation of the IPD into the JAUS architecture, set the IPD's unique component identification number at 99, and discussed the specifics of the IPD's functionality.

A Remotec Andros robot was chosen as the testing platform because of the indirect motion actuators located on the chassis and its complicated behavioral controls, namely stair-climbing. A complete JAUS controller was developed specifically to be used by the

Andros platform; an Operator Control Unit (OCU), Primitive Driver, Node Manager, Position System and Communicator components were coded for both the Rabbit RCM3200 microcontroller with the Dynamic C operating system and the Gateway Solo 3350 laptop with the Linux Redhat 8.0 operating system. The Rabbit was used as the Andros' JAUS controller, while the Solo 3350 was used as the OCU. The Intelligent Primitive Driver was applied to the Andros platform: the IPD's capability to control indirect motion actuators were mapped to the auxiliary tracks of the Andros and the behavioral control commands were mapped to the stair ascending and descending procedures. The proprietary Andros control stream was reverse engineered from the native Andros controller and then applied to the Primitive Driver and Intelligent Primitive Driver. As such, the Andros' JAUS controller was able to accept and interpret JAUS commands and incite the Andros to act upon these commands accordingly. The controller, as mounted on the Andros platform, can be seen in Figure 5-1.

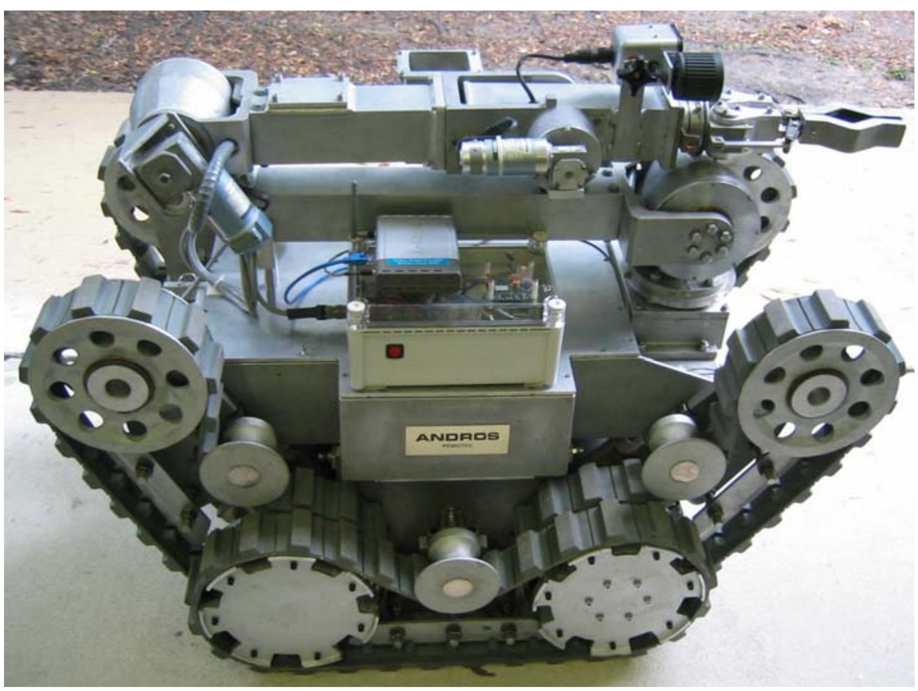

Figure 5-1: Controller mounted on Andros platform

The intelligence components of the Intelligent Primitive Driver were tested to ensure that they adequately performed the desired control function. A testing frame was constructed to constrain the motion of a sled housing one of the Sharp GP2D12 infrared object detectors so that the stair-counter assembly could be tested. From the data gathered from the system, it was shown that the negative second derivative impulses could be observed by the system and that they did indeed lag the detection of the outside stair edge by 0.25 seconds, as designed. The auxiliary track positioning system was tested to determine if the final angular placement of the auxiliary tracks coincided with the desired angular command. It was found that the average error value for the controller commanded halt angle was 0.5 degrees with a standard deviation of 0.3, whereas the average error value for the actual auxiliary actuator position was 3.4 degrees with a standard deviation of 2.2. The minimum allowable tolerance for this control system was +/- 1 degree. The heading alignment controller was tested to determine if the final heading of the Andros coincided with the desired heading command. It was found that the average error value for the controller commanded halt heading was 2.8 degrees with a standard deviation of 0.8, whereas the average error value for the actual halt heading of the Andros was 5.9 degrees with a standard deviation of 1.7. The minimum allowable tolerance for this control system was +/- 4 degrees. Both of these control systems performed as designed and returned favorable data when the limitations inherent to the Andros platform were taken into account.

The discrepancies between the DWT and plumb compass error in the auxiliary track positioning system and the final heading and compass heading error in the heading alignment controller are a direct manifestation of the limitations of the Andros system.

As discussed in Chapter 2, the maximum update rate of the Andros control stream is 5 Hz. In terms of autonomous computer controllers, 0.2 seconds is a relatively long amount of time. In both the auxiliary track positioning system and the heading alignment controller tests, the controller commanded a halt of system motion that was not acted upon until at least 0.2 seconds later. This fact translated into an average difference of 2.9 degrees for the auxiliary track positioning system and an average difference of 3.1 degrees for the heading alignment controller. Therefore, when the controller believes it has reached the desired command position and has thus stopped the motion of the Andros, the Andros is in actuality continuing to move. It is for this reason that the Andros stair motion behavior has been broken down into its component parts and these parts have been tested individually, instead of simply testing the Andros stair motion controller in its entirety; it is entirely within the realm of possibility that the Andros would incur a non-recoverable, fatal error when attempting to negotiate a staircase even though the controller recognized the error and attempted to correct it.

The Andros is able to negotiate a set of stairs when controlled by a human operator with a more reasonable expectation of a successful conclusion than with the JAUS controller for two reasons, though it is still a very difficult and complicated task. First, in terms of human control, 0.2 seconds is a relatively long amount of time. Second, the human operator has both control experience and predicate knowledge of the system. Therefore, the human operator can make predictive assumptions concerning the stair motion process that the JAUS controller is unable to make. For example, the human operator can take into account the type of stair to be traveled upon, the condition of the Andros' tracks, and the amount of traction that the Andros is exhibiting when actually on the stairs to make corrections to the stair motion process. With this incarnation of the IPD, these predictive capabilities are not present.

5.2 Future Work

To increase the controllability of the Andros-IPD system, upgrades need to be made to several of the subsystems. First, and most importantly, the Andros embedded controller should be updated to a more modern control scheme. The outdated controller and its 1200-Baud connection should be replaced by a modern microcontroller utilizing a secure Ethernet connection protocol, such as the Transmission Control Protocol (TCP). This replacement would solve the control problems, which currently inhibit the system. When making this update, the current drive motors and auxiliary track positioning motors should be replaced with motors that have integrated rotary encoders. This would serve two purposes. First, the encoders located on the driver motors could be utilized to add dead-reckoning capability to the current position system. With this, specific information concerning the total movement of each individual track could be collected and used for more exact vehicle placement. Second, the encoders on the auxiliary track positioning motors could replace the Draw Wire Transducers (DWT) currently used to acquire auxiliary track position. The DWT sensors work very well, but are susceptible to environmental conditions and hazards; the internal encoders would not have this potential problem. The Intelligent Primitive Driver should be modified to include adaptive control capabilities to the stair motion algorithms. Concepts of neural networks could be applied to the controller to give it the ability to learn from stair motion trials. This would allow the system to adapt the stair motion algorithms to work more efficiently on different types of stairs. For example, the controller could determine if an auxiliary track should be moved at a different point during the stair motion algorithm or change the correction speeds used to correct the Andros if it slips on the staircase.

Finally, the Intelligent Primitive Driver should be applied to other JAUS compliant platforms and further application testing should be performed. In doing this, the viability of the component could be completely explored and documented. After this, the component could be presented to the Working Group of the Joint Architecture for Autonomous Systems (JAUS) for the formal inclusion of the IPD into the Reference Architecture.

APPENDIX A
STANDARD JAUS MESSAGES

This Appendix contains information pertaining to the “Set Wrench Effort” and “Query Global Pose” standard JAUS messages, as well as a list of the core JAUS message set.

A.1 Query Global Pose

The command code for this message is defined as 0x2402 and causes the receiving component to respond with a “Report Global Pose” message. Table A-1 depicts the data field assignments for the “Query Global Pose” message [10].

Table A-1: “Query Global Pose” data field (used from JAUS: Reference Architecture Specification, pg. 93)

Field #	Name	Type	Units	Interpretation
1	Presence Vector	Unsigned Short	N/A	See Report Global Pose Message

A.2 Set Wrench Effort

The command code for this message is defined as 0x0405 and controls platform mobility actuators by mapping the six propulsive and six resistive elements that comprise a wrench command to the mobility actuators of a vehicle. Table A-2 depicts the data field assignments for the “Set Wrench Effort” message [10].

Table A-2: "Set Wrench Effort" data fields (from JAUS: Reference Architecture Specification, pg. 80)

Field #	Name	Type	Units	Interpretation
1	Presence Vector	Unsigned Short	N/A	See mapping table that follows.
2	Propulsive Linear Effort X	Short Integer	Percent	Scaled Integer Lower Limit = -100 Upper Limit = 100
3	Propulsive Linear Effort Y			
4	Propulsive Linear Effort Z			
5	Propulsive Rotational Effort X			
6	Propulsive Rotational Effort Y			
7	Propulsive Rotational Effort Z			
8	Resistive Linear Effort X	Byte	Percent	Scaled Integer Lower Limit = 0 Upper Limit = 100
9	Resistive Linear Effort Y			
10	Resistive Linear Effort Z			
11	Resistive Rotational Effort X			
12	Resistive Rotational Effort Y			
13	Resistive Rotational Effort Z			

Vector to Data Field Mapping for Above Command

Vector Bit	15	14	13	12	11	10	9	8
Data Field	R	R	R	R	13	12	11	10
Vector Bit	7	6	5	4	3	2	1	0
Data Field	9	8	7	6	5	4	3	2

"R" indicates that the bit is reserved.

A.3 JAUS Core Message Set

Table A-3 shows the core set of JAUS messages and their corresponding command codes [10].

Table A-3: JAUS core message set

Code	Description
0x01	Set Component Authority
0x02	Shutdown
0x03	Standby
0x04	Resume
0x05	Reset
0x06	Set Emergency
0x07	Clear Emergency
0x08	Create Service Connection
0x09	Confirm Service Connection
0x0A	Activate Service Connection
0x0B	Suspend Service Connection
0x0C	Terminate Service Connection
0x0D	Request Component Control
0x0E	Release Component Control
0x0F	Confirm Component Control
0x10	Reject Component Control

APPENDIX B
STAIR COUNTER MATHEMATICAL DEVELOPMENT

The ability to count the number of steps the Andros has passed is critical to the ability of the Intelligent Primitive Driver to perform its designed task. As discussed in Chapter 3, the current location on the stairs of the Andros is used to trigger controlled events during the stair climbing procedure. To ensure that this was done as accurately as possible, the optimum deployment angle for the infrared sensors needed to be determined so that the stair peaks and their corresponding second derivative impulses were easily observable by the system.

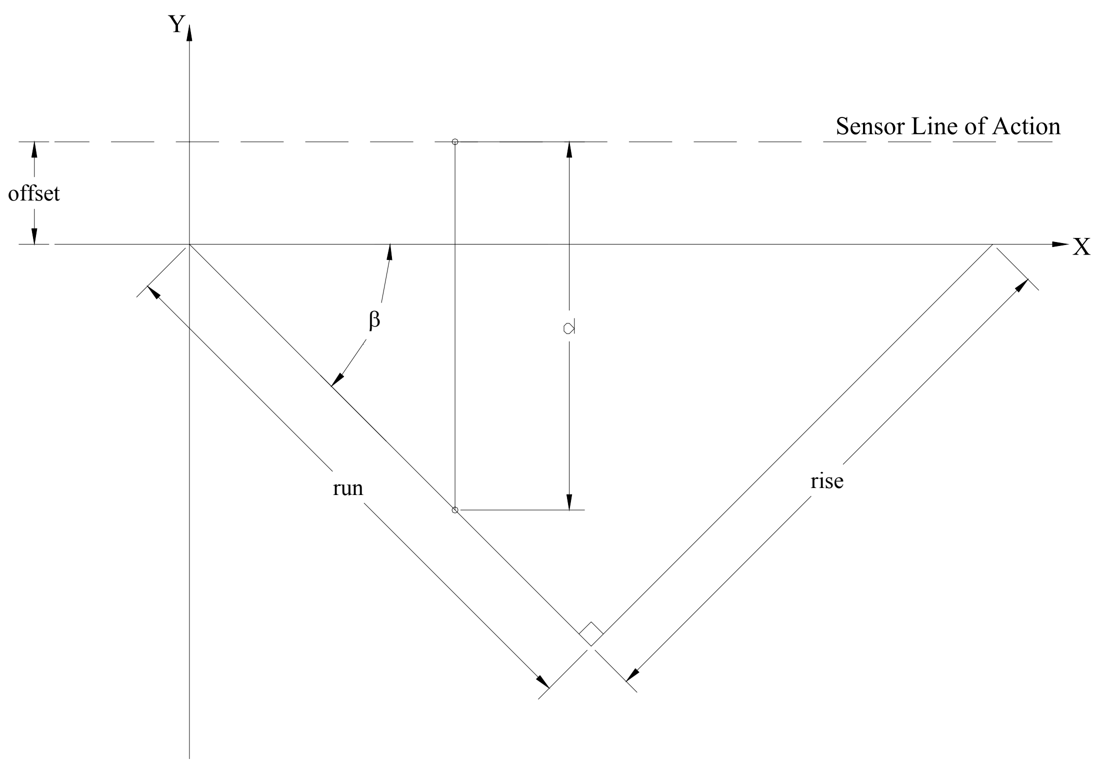

Figure B-1: Stair representation

The stair representation used for this development has been drawn relative to the plane of motion of the Andros, as shown in Figure B-1. This reference frame is appropriate as the sensor array moves along a path parallel to the plane of motion of the Andros and the measured distances are relative to this plane, and not to the ground plane. The single stair representation shown is defined by its rise, run, and the angle, β, which is determined through basic geometry and is constructed of two lines that are perpendicular to each other at their single point of intersection. As the stair representation is constructed of two linear equations, one for the rise and one for the run, two separate equations for the distance, **d**, must be developed. The distance, **d**, is the range distance measured from the infrared sensor to the profile of the stair.

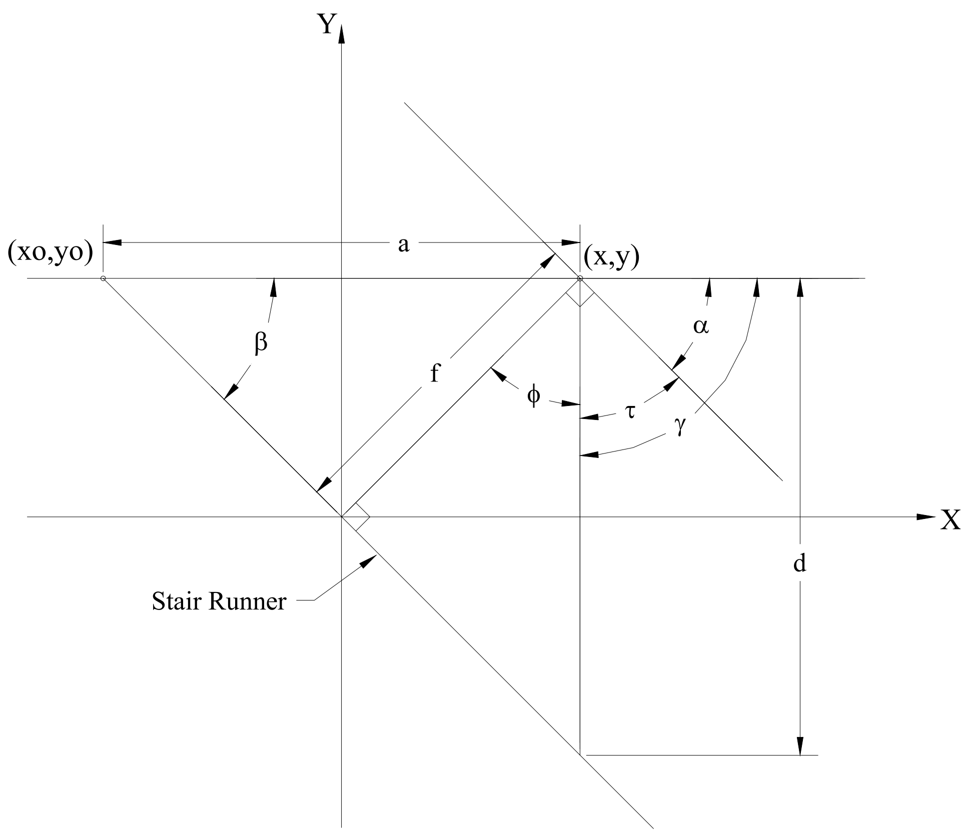

Figure B-2: Stair runner model

Figure B-2 shows the model used for the development of the equation for the distance, **d**, as it applies to runner of the stair. The first step in determining **d** is to find the point of intersection between the stair runner linear equation and the line of action of the sensor, (x_0, y_0). As the y-axis component of this point is constrained to be at y_0, x_0 can easily be found by solving the equation of the line. Using this as the point of origin, the distance, **a**, can be found to the point (x,y), which corresponds to the current location of the sensor. This distance is given as:

$$a = \sqrt{(x_0 - x)^2 + (y_0 - y)^2} \qquad \textbf{(B-1)}$$

Now that the distance, **a**, has been acquired, the perpendicular distance, **f**, can be found by the equation:

$$f = a \sin \beta \qquad \textbf{(B-2)}$$

Through the similar triangles concept, the angle, β, is shown on the two locations of Figure B-2. The angle, γ, is the user-defined angle of the infrared sensor. Therefore, the intermediate angle, τ, may be found, which then leads to the angle, ϕ, by the equation:

$$\phi = 90 - \gamma - \beta \qquad \textbf{(B-3)}$$

The distance, **d**, is now simply:

$$d = \frac{f}{\cos \phi} \qquad \textbf{(B-4)}$$

Figure B-3 shows the model used for the development of the equation for the distance, **d**, as it applies to riser of the stair. Again, the first step is to find the point of intersection, (x_0, y_0), between the stair riser linear equation and the line of action of the sensor. Again, as the y-axis value is constrained to be y_0, x_0 can be found by solving the equation of the line. Using this as the point of origin, the distance, **a**, can be found to the point (x,y), which corresponds to the current location of the sensor.

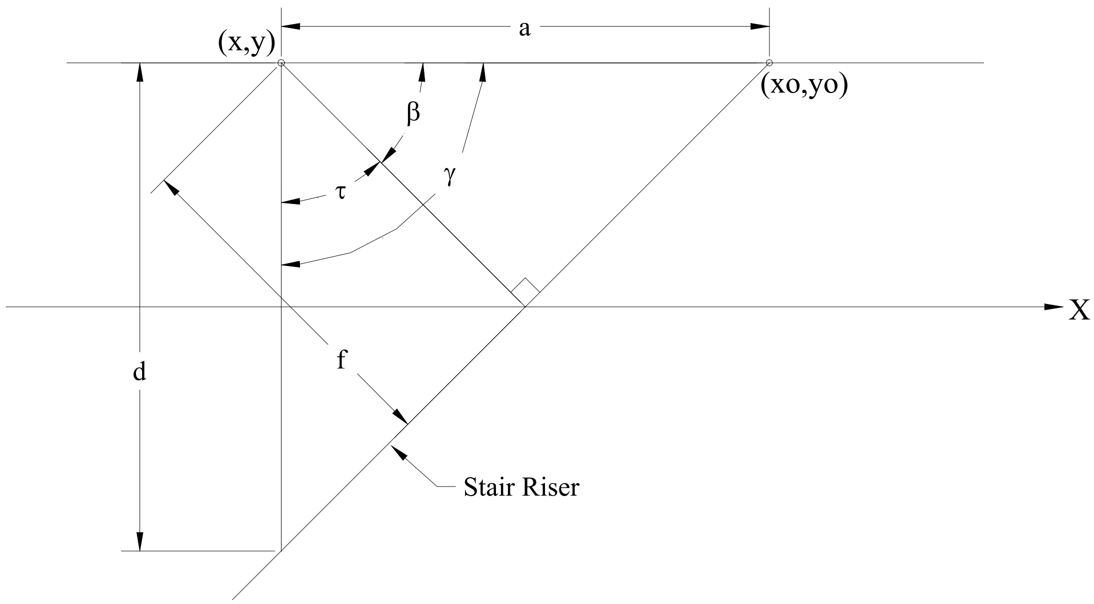

Figure B-3: Stair riser model

The equation for this distance is shown in Equation B-1. Through the similar triangles concept, the angle, β, is shown on the two locations of Figure B-3. The angle, γ, is the user-defined angle of the infrared sensor. Using γ and β, the angle, τ, may now be found by:

$$\tau = \gamma - \beta \tag{B-5}$$

The distance, **f**, is defined as:

$$f = a\cos\beta \tag{B-6}$$

Finally, the distance, **d**, may be found by:

$$d = \frac{f}{\cos\tau} \tag{B-7}$$

The two equations for the distance, **d**, and the stair data representation were placed into an Excel spreadsheet for analysis. From this, Figure B-4 was obtained. As shown in the figure, if the user-defined sensor angle, γ, is less than 90 degrees relative to the line of action of the sensors, the stair edge is skewed to the left and the risers are more pronounced over the range of the stair data. If γ is greater than 90 degrees to the line of action of the sensors, the stair edge is skewed to the right and the runners are more pronounced over the range of the stair data. Therefore, it was determined that γ should be 90 degrees, or perpendicular to the line of action of the sensors, as this case is neutral and shows no preference to either the risers or to the runners of the stair data. As the stair edges are more pronounced in the neutral case, so to will the second derivative impulses be more pronounced and more easily observed by the system.

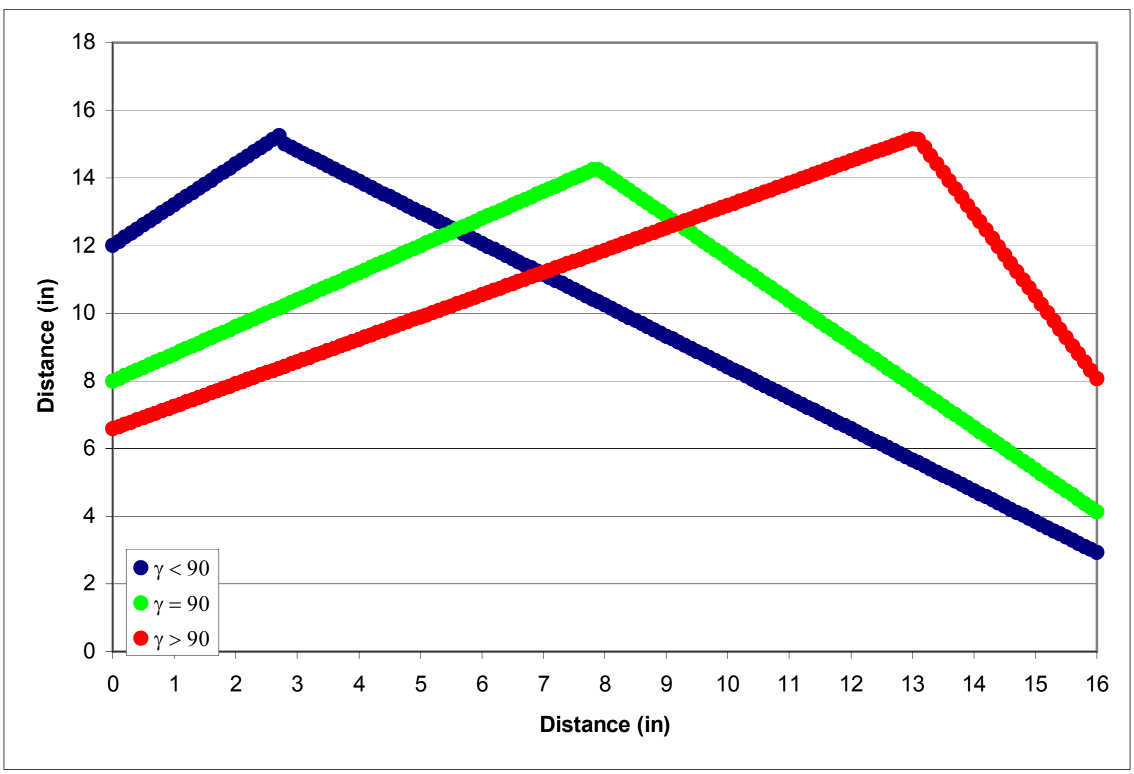

Figure B-4: Calculated distances, **d**, for varying value of γ

LIST OF REFERENCES

[1] Bayouth, M., Nourbakhsh, I., and Thorpe, C., "A Hybrid Human-Computer Autonomous Vehicle Architecture," Third ECPD International Conference on Advanced Robotics, Intelligent Automation and Control, 1997.

[2] Mustererkennung und Szenenanalyse (MESA) Research Group, 2003, "Semi-Autonomous Driving: Intelligent Cruise Control," MESA Research Group, http://www.neuroinformatik.ruhr-unibochum.de/PROJECTS/PATREC/projects/DAS/sadicc/sadicc_d.html, Feb 2003.

[3] Schlegel, N.,1997, "Autonomous Vehicle Control Using Image Processing," Virginia Polytechnic Institute and State University, http://scholar.lib.vt.edu/theses/public/ etd-283421290973280/etd.pdf, Feb 2003.

[4] Connell, J. and Viola, P., "Cooperative Control of a Semi-Autonomous Mobile Robot," Proc. of the 1990 IEEE International Conf. on Robotics and Automation (ICRA-90), 1990, pp. 1118-1121.

[5] Huntsberger, T., and Rose, J., "Behavior-based control for autonomous mobile robots," ROBOTICS2000, Albuquerque, NM, Mar 2000, pp. 299-305.

[6] Matthies, L., Xiong, Y., Hogg, R., Zhu, D., Rankin, A., Kennedy, B., Hebert, M., Maclachlan, R., Won, C., Frost, T., Sukhatme, G., McHenry, M., and Goldberg, S., "A Portable, Autonomous, Urban Reconnaissance Robot," The 6th International Conference on Intelligent Autonomous Systems, Venice, Italy, July 2000.

[7] Lewis, M., and Simo', L."Certain Principles of Biomorphic Robots," Autonomous Robots, volume 11, 2001, pp. 221-226.

[8] Talebi, S., M. Buehler, M., and Papadopoulos, E., "Towards Dynamic Step Climbing for a Quadruped Robot with Compliant Legs," Proceedings of the 3rd International Conference on Climbing and Walking Robots, Madrid, Spain, October 2000.

[9] Lauria, M., Piguet, Y. and Siegwart, R. "Octopus - An Autonomous Wheeled Climbing Robot," In Proceedings of the Fifth International Conference on Climbing and Walking Robots. Published by Professional Engineering Publishing Limited, Bury St Edmunds and London, UK, 2002.

[10] JAUS Working Group, 2002, “Joint Architecture for Unmanned Systems (JAUS): Reference Architecture Specification”, Version 3.0, Volume 2, The Joint Architecture for Unmanned Systems, http://www.jaugs.org, Feb 2003.

[11] Operations and Maintenance Manual for Andros Mark VI Robot System, Remotec, Inc., Oak Ridge, TN., (2001).

[12] Bock, R., 1998, “The Data Analysis BriefBook: Median Filter,” Forschungszentrum Juelich, http://ikpe1101.ikp.kfa-juelich.de/briefbook_data_analysis/node168.html, June 2003.

[13] Tanaka, K., An Introduction to Fuzzy Logic for Practical Applications, Springer, New York, New York, (1997).

BIOGRAPHICAL SKETCH

Peter M. Vinch, Jr. was born on October 29, 1976 in Lawrenceville, New Jersey to Peter and Nancy Vinch, Sr. He received his Bachelor of Science degree in mechanical engineering, graduating with honors, from the Rochester Institute of Technology in May of 2000. In August of 2000, he enrolled at the University of Florida and entered the master's program in the Department of Mechanical and Aerospace Engineering. Upon graduation from the University of Florida, he plans to work in the area of autonomous vehicle development.

University of Florida Graduate School
Electronic Thesis and Dissertation (ETD) Rights
and Permission

1. Enter and submit on line. 2. Print, sign, and submit to Graduate School Editorial Office

Student Name: ____

ID#: ____

Document Type: ________Master's Thesis ________Doctoral Dissertation

Document Title:

Student Agreement:

I hereby certify that I have obtained all necessary permission in writing for copyrighted material to be published in my thesis or dissertation. Further, I certify that I have obtained and attached hereto a written permission statement from the owner(s) of any copyrighted matter to be included in my thesis or dissertation, allowing distribution as specified below. Copies of all such permissions are maintained in my files.

I hereby grant to the University of Florida and its employees the nonexclusive license to archive and make accessible, under the conditions specified below, my thesis or dissertation in whole or in part in all forms of media, now or hereafter known. This is a license rather than assignments, and I, therefore, retain all other ownership rights to the copyright of the thesis or dissertation. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation.

In addition to the unrestricted display of the bibliographic information and the abstract, I agree that the above mentioned document be placed in the ETD archive with the following status (choose one of 1, 2, or 3) by checking the appropriate space below):

___ 1. Release the entire work immediately for access worldwide.

___ 2. Restrict access to the entire work to the University of Florida and all patrons of its libraries, including interlibrary sharing and release of doctoral dissertations to UMI. **[Please check appropriate time span.]** After (__1, __2, __3, __4, __5, __10) years release my work worldwide unless you receive in writing a request from me to continue restricted access. If I choose to release the work for worldwide access sooner, I will contact Library Archives, P.O. Box 117007, University of Florida, Gainesville, FL 32611.

___ 3. Secure the entire work for patent and/or proprietary purposes for a period of six months. During this period the copyright owner also agrees not to exercise her/his ownership rights, including public use in works, without prior authorization from the University of Florida. At the end of the six-month period, either I or the University of Florida may request an extension for an additional six months. At the end of the six-month secure period (or its extension, if such is requested), the work will be handled under option 1 above, unless I request option 2 in writing.

The undersigned agrees to abide by the statements above, and agrees that this approval form updates any and all previous approval forms submitted heretofore.

Signed: ______________________________ ____________________
(Student) (Date)

______________________________ ____________________
(Chair) (Date)

University of Florida Graduate School
Electronic Thesis and Dissertation (ETD) Signature Page

Student's Name: ____

This document has been reviewed and accepted by the student's supervisory committee.

Professor's name & title including department	Professor's signature
______________________________Chair	______________________________
______________________________	______________________________
______________________________	______________________________
______________________________	______________________________
______________________________	______________________________
______________________________	______________________________
______________________________	______________________________

Month and Year of Graduation: ______________________________

College Dean: ______________________________
(when required by college)

Graduate Dean: ______________________________